UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

ROMÁRIO ANTÔNIO MAURÍCIO JÚNIOR

CONSTRUÇÃO DE UM PROCESSO BASEADO NA MATRIZ DE RASTREABILIDADE DE TESTES APLICADO À PRIORIZAÇÃO DE CASOS DE TESTES NA VALIDAÇÃO DE FUNCIONALIDADES DE SOFTWARE

CRICIÚMA

2015

ROMÁRIO ANTÔNIO MAURÍCIO JÚNIOR

CONSTRUÇÃO DE UM PROCESSO BASEADO NA MATRIZ DE RASTREABILIDADE DE TESTES APLICADO À PRIORIZAÇÃO DE CASOS DE TESTES NA VALIDAÇÃO DE FUNCIONALIDADES DE SOFTWARE

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. MSc. Gustavo Bisognin.

CRICIÚMA

2015

ROMÁRIO ANTÔNIO MAURÍCIO JÚNOR

CONSTRUÇÃO DE UM PROCESO BASEADO NA MATRIZ DE RASTREABILIDADE DE TESTES APLICADO À PRIORIZAÇÃO DE CASOS DE TESTES NA VALIDAÇÃO DE FUNCIONALIDADES DE SOFTWARE

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, 25 de junho de 2015.

BANCA EXAMINADORA

Prof. MSc. Gustavo Bisognin (UNESC) - Orientador

Profa. MSc. Ana Claudia Garcia Barbosa (UNESC)

Prof. Esp. Gilberto Vieira da Silva (UNESC)

Dedico este trabalho a meus pais e a minha irmã que estiverem sempre ao meu lado incentivando nesta empreitada.

AGRADECIMENTOS

Agradeço primeiramente e principalmente a Deus, que me deu forças e coragem para enfrentar esse grande desafio.

Aos meus pais Romário e Izolete que sempre se preocuparam em me dar uma educação de qualidade e que nunca deixaram de acreditar na minha capacidade para vencer os desafios.

A minha irmã Denise, que sempre dedicou um pouco do seu tempo e do seu conhecimento para me auxiliar na realização desse trabalho.

Agradeço ainda, o meu orientador, Prof. MSc Gustavo Bisognin, que me auxiliou, compartilhando o seu conhecimento.

A empresa Domínio Sistemas, agora parte da multinacional Thomson Reuters, pelos desafios e pela oportunidade de trabalhar na área de testes. Aos meus colegas de trabalho, principalmente ao Cleverson Reinert, gerente de teste de software, que me incentivou e aconselhou na realização deste trabalho.

E não menos importante, agradeço a todos os meus professores de Ciência da Computação, da UNESC e aos meus amigos Ane Assunção, Bruno Amaral, Diogo Anphiloquio, Felipe Vieira, Filipe Pizzolo, Gabriela Schaukoski, Guilherme de Souza, Guilherme Zocche, Jadson Frassetto, Leonardo Bocianoski, Lucas dos Santos, Moisés Coral, Renato Macarini e Zaquel Matiola.

Enfim, agradeço a todos que contribuíram direta ou indiretamente para que eu concluísse este trabalho.

“Errar é humano, encontrar um erro é divino.”

Robert Dunn

RESUMO

Atualmente é essencial a realização de testes nos sistemas, para garantir a sua qualidade. Adequado a esse cenário, este estudo apresenta a criação de uma técnica para priorização dos casos de testes feitos no sistema para ampliar a validação das funcionalidades do software. O trabalho começa com pesquisa e conhecimento sobre definições da engenharia de software, como por exemplo, requisitos de software, suas características, seus benefícios, sua estrutura, assim como conceitos pertinentes aos requisitos funcionais e requisitos não funcionais ou de qualidade. Após isso, os testes de software, baseados em requisitos, são analisados, investigando a cobertura funcional e a derivação funcional com evidência nas técnicas de derivação funcional formal e informal, salientando, a técnica de conhecimento e experiência do especialista de teste. Na sequência, o ciclo de vida também é estudado apresentando alguns modelos e suas estruturas, bem como os componentes importantes para essa estruturação como as fases de integração, fase de sistema e fase de regressão. Dando continuidade, a pesquisa aborda a matriz de rastreabilidade (TRM) e sua estruturação, o princípio de Pareto e a matriz de priorização GUT que serão utilizadas conjuntamente para obter uma melhor avaliação das prioridades. E finalizando a fundamentação teórica são demonstradas as etapas de planejamento de teste juntamente com a definição do plano e dos casos de teste. Na fase prática, ocorre a seleção do software IRPF2015 - Declaração de Ajuste Anual, Final de Espólio e Saída Definitiva do País - da Receita Federal para ser testado. Utilizando a derivação com base no conhecimento e experiência do especialista, o trabalho contempla o levantamento dos casos de teste e sua inserção na matriz de rastreabilidade. Nesta fase, os casos de testes recebem uma classificação com base no conceito da matriz de GUT e são selecionados utilizando o princípio 80/20 de Pareto. Por último, os casos de testes são executados e a planilha com a matriz de rastreabilidade é preenchida conforme o resultado apresentado para cada caso de teste efetuado.

Palavras-chave: PRIORIZAÇÃO DE CASOS DE TESTE. TESTE DE SOFTWARE. DERIVAÇÃO FUNCIONAL. QUALIDADE DE SOFTWARE. MATRIZ DE RASTREABILIDADE.

ABSTRACT

Currently is essential the realization for tests on systems, to ensure their quality. Suitable for this scenario, this study presents the creating of a technique for priorization of tests cases made on the system to extend the validation of the functionality of the software. The work begins with research and knowledge about definitions of the software engineering, as for example, software requirement, your characteristics, your benefits, your structure, as well pertinent concepts to requirements functional and nonfunctional requirements or quality. After that, the tests of software, based on requirements, are analyzed, investigating the functional coverage and the functional derivation with evidence in formal functional derivation techniques and informal, stressing, the technique of the knowledge and experience of test specialist. On sequence, the life cycle is also studied presenting models and their structures, as well as the important components for this organization as integration phases, phase system and regression phase. Continuing, the research approach the traceability rastreability matrix (TRM) and your structuration, the Pareto principle and the matrix of priorization GUT, to be used together to get a better assessment of priorities. And finishing the theoretical fundamentation are demonstrated test planning stages along with the definition of the plan and of the test cases. In the practical phase, occur the selection of the software IRPF2015 - Annual Adjustment Statement, last assets and Final Departure Country – of the Federal Revenue to be tested. Using the derivation based on the knowledge and experience of test specialist, the work includes the lifting of test cases and their inclusion in the traceability matrix. In this phase, the test cases are given a rating based on the concept of matrix GUT and are selected using the principle 80/20 of the Pareto. Lastly, the test cases are executed and the spreadsheet with the traceability matrix is filled with the result presented for each performed test case.

Palavras-chave: PRIORITIZATION OF TEST CASES. TEST OF SOFTWARE. DERIVATION FUNCTIONAL. QUALITY OF SOFTWARE. TRACEABILITY MATRIX.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ATIFS	Ambiente de Testes baseado em Injeção de Falhas por Software
DERS	Documento de Especificação de Requisitos de Software
GOT	Guia Operacional de Teste
GUT	Gravidade, Urgência e Tendência
IEEE	Institute of Electrical and Electronics Engineers
INPE	Instituto Nacional de Pesquisas Espaciais
NFR	Requisitos Não Funcionais ou de Qualidade
PBQP	Programa Brasileiro de Qualidade e Produto em Software
RBT	*Requirements-Based Testing*
TRM	Traceability Rastreability Matrix
UNESC	Universidade do extremo sul catarinense

SUMÁRIO

1 INTRODUÇÃO

A utilização dos testes nos sistemas é de suma importância para garantir a qualidade do mesmo e a satisfação do cliente em utilizar um software que possua poucos erros e que não influencie negativamente em seu trabalho.

Segundo Dias Neto (2007), teste de software é o método de execução de um sistema para definir se ele alcançou suas especificações e funcionou de modo correto no ambiente para o qual foi planejado. Nesse processo, têm-se como objetivo encontrar a maior quantidade de falhas possíveis, visando maior confiança no produto fornecido ao cliente.

Conforme Pressman (2006), o software está em constante evolução e apresentará defeitos durante toda a sua existência, devido às frequentes alterações realizadas no sistema para se adequar ao que o cliente necessita, ou seja, quanto mais alterações são realizadas no sistema, maior será a probabilidade de surgirem novas falhas, que acarretam na queda da qualidade do software. O governo brasileiro preocupado com a qualidade no desenvolvimento de softwares criou o projeto Programa Brasileiro de Qualidade e Produto em Software (PBQP), que incentiva a criação de melhores práticas para desenvolver sistemas (BRASIL, 2008).

O tempo destinado a execução de testes é curto. Para que, isso não resulte em uma queda na qualidade de um sistema são utilizados critérios para derivar os casos de testes que serão executados (DIAS NETO, 2014). Existe o critério de particionamento em classes de equivalência, que de acordo com Rocha et al (2001) tem por objetivo diminuir a quantidade de casos de teste, fazendo a seleção de apenas um caso de teste de cada classe, considerando que todos os elementos de uma classe devem possuir o mesmo comportamento. Outro critério que pode ser utilizado é o critério de análise do valor limite, que conforme Pressman (2005) um grande número de erros tende a ocorrer nos limites dos valores do domínio de entrada de cada classe. Existe também o critério Grafo de causa-efeito, que segundo Rocha et al (2001), verifica o efeito combinado de dados de entrada, identificando as causas e os efeitos para então tomar a decisão de quais casos de testes serão executados.

Além desses, de acordo com Dias Neto (2014), outros critérios podem ser utilizados dependendo da necessidade. Dentre estes critérios estão: teste de desempenho, teste de usabilidade, teste de carga, teste de stress, teste de confiabilidade e teste de recuperação.

Além da derivação dos casos de teste, a rastreabilidade de testes, que segundo Gotel e Finkelstein (1995, tradução nossa) é a habilidade de descrever e acompanhar um requisito ao longo de um ciclo de vida, tanto na direção das suas fontes (pessoas, ideias, artefatos) como na direção dos artefatos que o sucedem é indispensável para obter um software com qualidade (CLELAND-HUANG, 2003; GOTEL; FINKELSTEIN, 1995; RAMESH; JARKE, 2001).

Sendo assim, esta pesquisa propõe a criação de uma técnica para priorizar os casos de testes feitos no sistema. Considerando-se uma tela de login, pode-se perceber que existem muitas variações de testes para serem executados para garantir que serão atendidos 100% dos requisitos dessa tela. Porém, isso seria inviável, devido à quantidade de tempo que seria consumido. Sendo assim, objetiva-se reduzir o tempo total gasto para executar os casos de testes, garantindo a qualidade do software.

1.1 OBJETIVO GERAL

Construir um processo baseado na matriz de rastreabilidade de testes como apoio a priorização da execução de casos de teste na fase de integração do ciclo de vida de testes de software.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa consistem em:

a) avaliar o conceito de rastreabilidade de testes de software;
b) analisar técnicas de priorização;
c) pesquisar sobre a construção de casos de testes utilizando o método de derivação de conhecimento e experiência do especialista de teste;
d) elaborar a matriz de rastreabilidade de testes;

e) construir o método de priorização e associar com a matriz de rastreabilidade;
f) validar o processo de criação da técnica de priorização;
g) modelar e publicar o processo.

1.3 JUSTIFICATIVA

Uma falha é um imprevisto que acontece quando as funcionalidades de um software não condizem com o objetivo real que ele deveria exercer (AVIZIENIS et al, 2004, tradução nossa).

O mesmo autor ainda afirma que um erro é a situação em que as funcionalidades de um sistema não equivalem à definição funcional do produto ou quando a definição não estabelece o objetivo do software de modo correto.

As falhas que surgem ao longo da utilização de um software, normalmente são acarretadas pela implementação incorreta da funcionalidade ou falha na sua definição. Softwares com este problema geram prejuízos como refazer o trabalho, extravio de dados, atrasos em operações relevante, entre outros.

Yourdon (1990) assegura que nenhum sistema é absolutamente protegido e isento de falhas. Isso ocorre, pois muitas funcionalidades não são muito utilizadas e consequentemente certos erros de maneira nenhuma serão encontrados, ou seja, o sistema conserva-se indefinidamente com defeito, não permitindo o seu reparo.

A criação de uma técnica para priorizar os casos de testes feitos em um sistema deve ser feita sempre pensando na qualidade de software. O emprego de tal estudo permitirá a construção de um processo baseado na matriz de rastreabilidade de testes aplicado à priorização de casos de testes. Tendo como objetivo validar as funcionalidades, buscando a qualidade.

Além disso, por ser a área de atuação profissional do pesquisador, trará maiores conhecimentos em relação aos conceitos utilizados no processo de teste e em relação aos casos de teste criados, avaliados, priorizados e executados na empresa em que atua, fazendo com que cresça profissionalmente.

O tempo utilizado para execução dos casos de teste será um dos principais fatores que influenciarão na priorização dos casos no processo de teste, porém necessitará que seja assegurada a qualidade do software, ou seja, este não é o fator mais relevante, já que não se pode deixar de executar um teste crítico em função do fator tempo.

A criação da matriz de rastreabilidade fará com que sejam controlados os casos de testes executados no processo. Além disso, serão documentados todos os detalhes do planejamento de teste, como a informação se o caso encontrou alguma falha ou não, se o teste foi planejado, caso sim, se foi também executado, informações do testador responsável e quando foi realizada esta execução.

1.4 ESTRUTURA DO TRABALHO

Os temas abordados pelo trabalho foram distribuídos em 8 capítulos, sendo o primeiro composto pela introdução, objetivo principal, objetivos específicos, justificativa e a própria estrutura do trabalho, aqui relatada. Quanto os demais capítulos, os assuntos abordados foram:

a) **capítulo 2, Requisitos de software:** conceitua as características do requisitos de software, além de definir os requisitos funcionais e não funcionais;
b) **capítulo 3, Teste de software baseado em requisitos:** este capítulo tem como objetivo detalhar o processo de teste de software baseado em requisitos, abordar o método de cobertura funcional e descrever a derivação funcional, além de conceituar algumas de suas técnicas;
c) **capítulo 4, Ciclo de vida de teste:** apresenta um estudo do ciclo de vida de teste e de suas principais fases;
d) **capítulo 5, Matriz de rastreabilidade:** expõe o conceito de matriz de rastreabilidade, suas características e a sua importância. Além disso, aborda o conceito da matriz de priorização GUT e a definição do Princípio de Pareto;

e) **capítulo 6, Planejamento de teste:** contém a definição do planejamento de teste e apresenta as propriedades para a definição do plano de teste e dos casos de teste;

f) **capítulo 7, Trabalhos correlatos:** expõe o resultado dos estudos efetuados em 4 trabalhos correlatos levantados na etapa da fundamentação teórica, os quais possuem os seguintes títulos:

- Desenvolvimento de uma Metodologia Baseada na Automação do Processo de Testes de Software como Estratégia para a Garantia da Cobertura Funcional (DAMIANI, 2012),
- A Importância do Processo de Teste para a Qualidade do Software (CAVALCANTI, 2009),
- Utilização de Testes Automatizados para Ampliar a Cobertura Funcional na Manutenção de Software (REINERT, 2012),
- Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo (CRESPO et al, 2012);

g) **capítulo 8, Construção de um processo baseado na matriz de rastreabilidade de testes aplicado à priorização de casos de testes na validação de funcionalidades de software:** aborda o trabalho desenvolvido, introduzindo conceitos que levaram à realização do trabalho e apresentando a metodologia colocada em prática. O último subcapítulo retrata os resultados que o trabalho forneceu perante as finalidades pretendidas. Nos demais subcapítulos são especificados os estágios que integraram a metodologia, que são:

- Levantamento Bibliográfico,
- Estudo e seleção do software para realizar a execução dos casos de teste,
- Criação dos casos de teste de acordo com as três técnicas de derivação mais utilizadas,
- Classificação dos casos de teste com a matriz de priorização GUT,
- Priorização dos casos utilizando Pareto,

- Criação da matriz de rastreabilidade,
- Execução dos casos de teste.

2 REQUISITOS DE SOFTWARE

Requisito é uma condição ou exigência indispensável para resolver um problema ou alcançar uma determinada finalidade.

O requisito de software, por sua vez, é um campo dentro da engenharia de software que descreve as funcionalidades que deverão ser providas por este sistema e quais serão suas restrições operacionais e restrições satisfeitas no processo de desenvolvimento (SOMMERVILLE, 2007). Esses requisitos devem ser levantados juntamente com o cliente, porém é importante estar atento a isso, pois mesmo o requisito sendo definido pelo cliente, nem sempre a funcionalidade que ele quer é o que o negócio realmente necessita. Além disso, deve-se ressaltar que os requisitos fornecerão uma referência para a validação do software.

Uma boa especificação desses requisitos de software é o Documento de Especificação de Requisitos de Software (DERS) que deve possuir características como (figura 1):

Figura 1 – Características da DERS

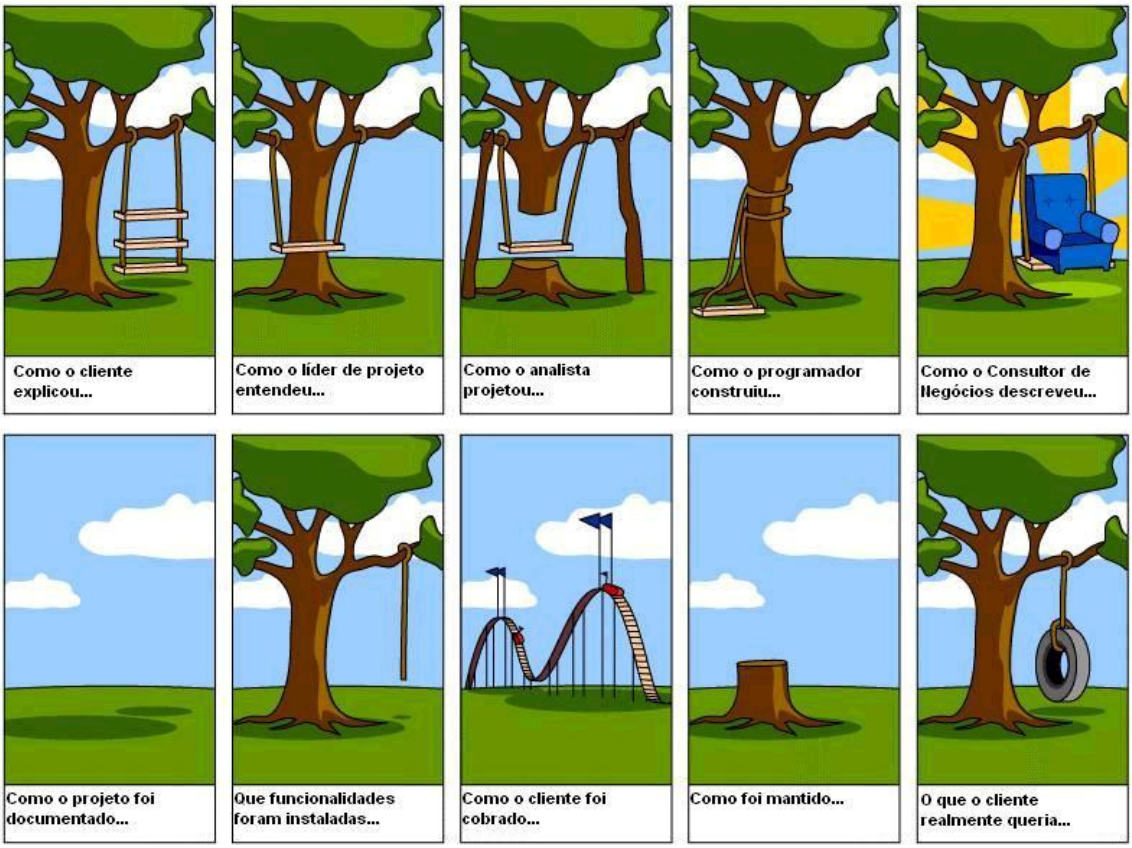
Correta – cada requisito expresso deve ser encontrado também no software;
Clara e não ambígua – cada requisito deve possuir somente uma interpretação;
Completa – Deve incluir elementos como: reconhecimento e tratamento de todos os requisitos do sistema, especificação do comportamento para os valores de entradas válidos e inválidos e referência das figuras, tabelas e diagramas e definição dos termos e unidades de moeda;
Consistente – Nenhum requisito individual estiver em conflito;
Superior pela importância e/ou estabilidade – identificar a importância dos requisitos através do grau de estabilidade ou grau de necessidade;
Verificável – quando existe a possibilidade de checar que o produto de software encontra o requisito;
Modificável – quando sua estrutura e estilo permitem qualquer mudança;
Rastreável – quando a origem de cada requisito for clara e facilitar a referência de cada requisito afetado por uma modificação, ou seja, um DERS reconstituível.

Fonte: Adaptado de IEEE[1] (1998, tradução nossa).

[1] Institute of Electrical and Electronics Engineers (Instituto de Engenheiros Eletricistas e Eletrônicos)

A alta complexidade das áreas de processo, gerenciamento e desenvolvimento de requisitos se deve, principalmente pelo alto nível de subjetividade do negócio. A especificação ou identificação incompleta das características do software podem acarretar em problemas sérios no desenvolvimento, funcionamento e satisfação do cliente em relação à solução proposta, podendo causar a não execução ou continuidade do projeto. Além disso, os requisitos podem gerar impacto negativo na construção do software, como atrasos, retrabalho e consequentemente um aumento no custo do projeto (DIAS; ARAÚJO, 2010).

Figura 2 – Demonstração de uma especificação de requisito de software que não foi bem estruturada



Fonte: Pressman (2009).

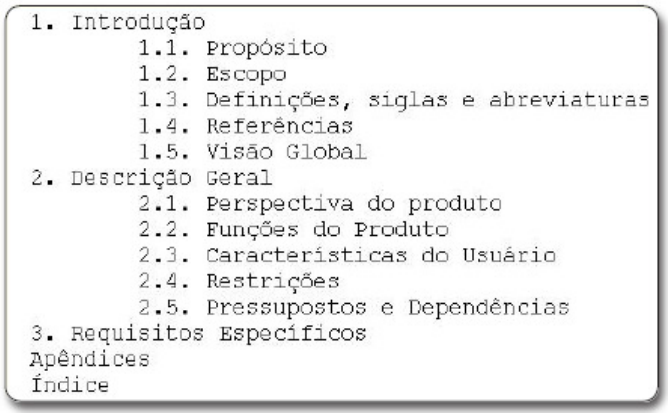
Para tornar a construção dos requisitos correta é importante o conhecimento e a utilização de técnicas. Uma dessas técnicas é o DERS, que serve para que clientes e desenvolvedores consigam chegar a um consenso

em relação ao que o cliente realmente necessita. Ainda segundo o IEEE (1998, tradução nossa), o DERS provém benefícios como:

a) **acordo entre clientes e desenvolvedores sobre a funcionalidade do software:** para que o sistema possa atender as necessidades do cliente;
b) **prevenção de retrabalho:** revisar cuidadosamente os requisitos da DERS para corrigir possíveis problemas antes do início do desenvolvimento;
c) **base para estimativa de custos e prazos:** pode ser usado para aprovação dos custos e prazos;
d) **base para validação e verificação:** permite planejar a validação e verificação (testes);
e) **facilita a manutenção do software:** dependendo da alteração será necessária a alteração no DERS, porém o mesmo servirá como base para essas alterações.

O IEEE organizou um documento com as práticas recomendadas para a criação de uma DERS. A figura 3, mostra como deve ser estruturada uma DERS.

Figura 3 – Estrutura de uma DERS



Fonte: Adaptado de IEEE (1998, tradução nossa).

Dentre os itens da estrutura acima, “Requisitos específicos” é o principal item, pois é o que abrange a definição dos requisitos funcionais e não funcionais.

2.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais são relacionados às funcionalidades do sistema, bem como a seu comportamento diante de entradas específicas e sua reação em determinadas situações. Além disso, alguns requisitos funcionais podem determinar claramente o que o sistema não deve realizar (LAPLANTE, 2014, tradução nossa).

Os requisitos funcionais expõem em detalhes as funções do sistema como as entradas e saídas, exceções, entre outros. Vale lembrar, que os requisitos dependem do tipo de sistema a ser implementado, dos usuários destinados a utilizar o software e da forma que serão redigidos pela organização (SOMMERVILLE, 2007).

Com essas definições, compreendemos que são exemplos de requisitos funcionais em um software:

a) cadastrar fornecedores;
b) fazer análise de crédito;
c) fazer uma transação com o banco;
d) imprimir relatório;
e) permitir a baixa automática do estoque quando um produto for vendido.

Segundo Engholm Junior (2010) são tipos de requisitos funcionais:

a) **casos de uso:** que especificam o comportamento do software a ser desenvolvido, sem indicar a maneira com que o comportamento será implementado;
b) **funções:** que especificam o conjunto de entradas, seu comportamento e saídas;
c) **regras de negócio:** que especificam as peculiaridades das funcionalidades a serem implementadas;
d) **interfaces externas:** que especificam a comunicação entre o software em desenvolvimento com outros softwares;
e) **interfaces internas:** que especificam a comunicação entre os elementos da arquitetura.

Segundo Sommerville (2007), as especificações dos requisitos funcionais de um sistema devem ser:

a. **Completas:** todas as funcionalidades requeridas pelo usuário devem ser definidas;
b. **Consistentes:** não devem possuir definições incoerentes.

Porém, sabe-se que é praticamente impossível alcançar as características acima em softwares grandes e complexos.

O subcapítulo seguinte detalhará os requisitos não funcionais, apresentando sua definição, demonstrando algumas características e expondo alguns exemplos.

2.2 REQUISITOS NÃO FUNCIONAIS OU DE QUALIDADE

Os Requisitos Não Funcionais ou de Qualidade (NFR) são aqueles não relacionados às utilidades providas pelo software. Segundo Pressman (2001), esses requisitos podem ser vinculados às restrições sobre serviços ou funções do software, como restrições de tempo, do processo de desenvolvimento, de produções, entre outros. Logo, são atribuídos a propriedades do sistema como confiabilidade, tempo de resposta, espaço em disco, desempenho e outras peculiaridades de qualidade (SOMMERVILLE, 2008).

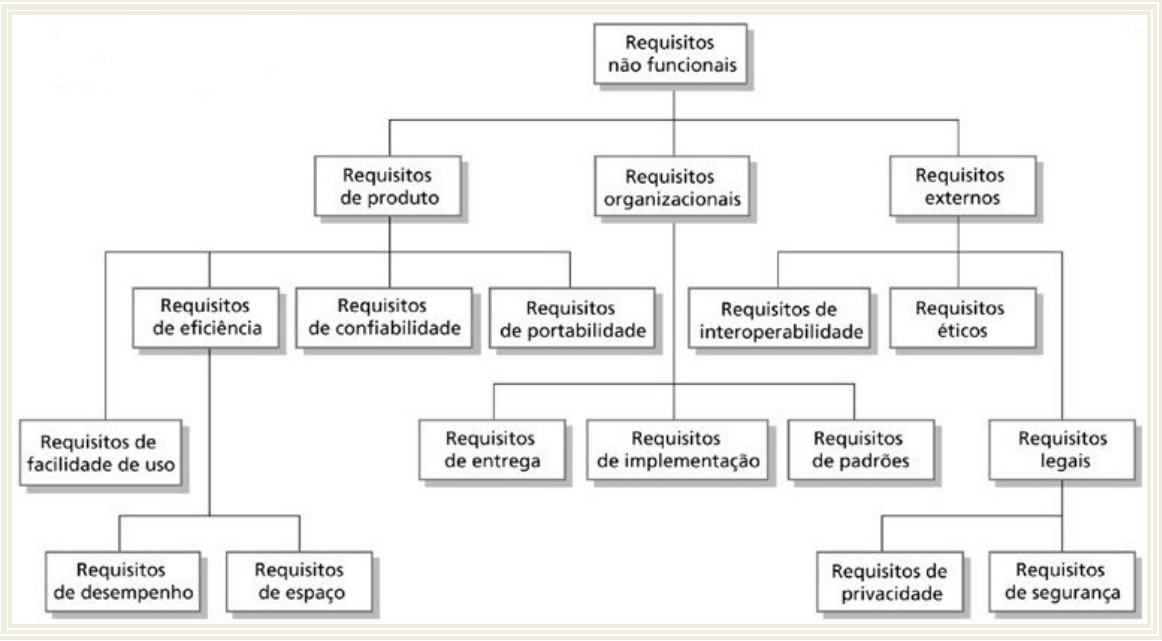
O autor afirma ainda que os requisitos não funcionais na maioria dos casos são mais essenciais que os requisitos funcionais, pois o não cumprimento de um requisito funcional pode comprometer parte do sistema, enquanto que uma falha no cumprimento de um requisito não funcional pode inutilizar o software. Por exemplo, se os requisitos de desempenho e confiabilidade de um sistema de tráfego aéreo possuir uma falha, ele não operará corretamente e consequentemente não poderá ser atestado como seguro para orientar e monitorar o trajeto de aeronaves.

Ainda segundo Sommerville (2008), os requisitos de qualidade não tratam somente do sistema de softwares a ser implementado, mas também podem delimitar os métodos utilizados para desenvolver o software. Sendo assim, podemos afirmar que pode incluir desde uma especificação de modelos de qualidade a ser utilizada no processo até a indicação de ferramentas a serem empregadas.

Conforme a figura 4, podemos visualizar a forma em que Sommerville (2008) classificou os requisitos não funcionais. Nesse diagrama, percebemos que os requisitos não funcionais podem resultar de peculiaridades indispensáveis do software (requisitos de produtos), da instituição que desenvolve o sistema (requisitos organizacionais) e de fatores externos.

Segundo Sommerville (2008), os tipos de requisitos não funcionais são:

Figura 4 – Tipos de requisitos não funcionais



Fonte: Sommerville (2008).

a. **requisitos de produto:** que especificam o comportamento do produto (sistema). Referem-se aos atributos de qualidade que o software deve possuir, como facilidade de uso, eficiência, confiabilidade e portabilidade;
b. **requisitos organizacionais:** que são originados de políticas e métodos seguidos nas organizações do cliente e do desenvolvedor. O sistema deve ser especificado conforme um processo-padrão da empresa. Referem-se a atributos de entrega (definindo a entrega do produto com sua respectiva documentação), de implementação (definindo a linguagem de programação ou método de projeto a ser utilizado) e de padrões de processo;

c. **requisitos externos:** que envolvem fatores externos ao sistema e ao processo de desenvolvimento. São abrangidos por esse item os requisitos de interoperabilidade (necessidade de interação com outros sistemas), requisitos legais (que garantem que o sistema está funcionando dentro dos padrões e da lei) e os requisitos éticos (para certificar que será aprovado pelos seus usuários).

Além dos requisitos funcionais e dos não funcionais, alguns autores ainda consideram outro requisito que é chamado de requisito de domínio. Segundo Laplante (2014, tradução nossa), requisitos de domínio são relacionados a características resultadas do domínio da aplicação e refletem as características do mesmo. Esse requisito pode consistir de novos requisitos funcionais, de restrições de requisitos funcionais existentes ou podem estabelecer de que forma um determinado cálculo deverá ser realizado.

No capítulo seguinte será abordado o processo de teste de software baseado em requisitos, detalhando seus conceitos e demais características.

3 TESTE DE SOFTWARE BASEADO EM REQUISITOS

O teste de software, segundo Gustafson (2002), consiste na execução de um conjunto de casos de teste com o objetivo de indicar que o software possui erros ou que o sistema não está executando suas funções ou seus requisitos conforme projetado inicialmente.

O teste de software baseado em requisitos, do inglês *Requirements-Based Testing* (RBT), por sua vez, é um processo cujo objetivo é executar determinadas entradas - casos de teste - e testar o software avaliando se as respostas geradas pelo sistema estão em concordância com o previsto no DERS (DELAMARO et al, 2007).

Segundo Magyoridi (2003, tradução nossa), a metodologia RBT é um procedimento que possui 12 passos, sendo eles:

a) **validação dos requisitos versus objetivos:** compara os objetivos (critérios de sucesso para o projeto) descritos no início do projeto com os requisitos que descrevem o que deverá ser entregue, caso um requisito não atinja seus objetivos, o mesmo não pertencerá ao escopo do projeto;
b) **validação dos casos de uso em relação aos requisitos:** a documentação dos requisitos com a utilização de casos de uso é adotada em algumas organizações. O caso de uso é vastamente utilizado para constatar e registrar requisitos. Os requisitos individuais devem ser capazes de satisfazer todos os cenários de casos de uso, senão serão considerados como incompletos;
c) **realização de revisão inicial de ambiguidade:** técnica para apontar e suprimir palavras, frases ou construções ambíguas. Não é uma revisão do conteúdo dos requisitos. Essa revisão gera uma qualidade maior na definição dos requisitos analisados pela equipe do projeto;
d) **realização de revisão do especialista de domínio:** os requisitos são revistos com o objetivo de corrigi-los e/ou completá-los;
e) **criação de gráfico de causa-efeito:** os requisitos são traduzidos em um gráfico de causa-efeito, proporcionando benefícios como:

resolução do problema com *aliases* (usando termos diferentes para a mesma causa-efeito), esclarecimento das regras de precedência entre os requisitos (as causas obrigatoriamente devem satisfazer aos efeitos), esclarecimento de informações subentendidas (tornar as informações compreensíveis para a equipe do projeto) e demonstra a integração das causas e efeitos;

f) **verificação de consistência lógica realizada e casos de teste projetados:** identificação de quaisquer erros de lógica no gráfico de causa-efeito. A saída é um conjunto de casos de teste 100% correspondentes às funcionalidades dos requisitos;
g) **revisão dos casos de teste pelos autores dos requisitos:** se houver algum problema com os casos de testes, os requisitos vinculados a ele podem ser corrigidos e refeitos;
h) **validação dos casos de teste com os usuários e especialistas de domínio:** os usuários e especialistas de domínio obtém uma melhor compreensão do sistema que será entregue;
i) **revisão dos casos de teste pelos desenvolvedores:** ao realizar esta tarefa os desenvolvedores assimilam o que será testado e obtém uma melhor compreensão do que deverá entregar;
j) **usando os casos de teste na revisão do projeto:** os casos de testes ratificam os requisitos como uma série de causas-efeito. Os casos de testes podem ser utilizados para confirmar que o projeto é robusto o suficiente para satisfazer os requisitos. Caso o projeto não cumpra os requisitos, os mesmos são impraticáveis ou necessitam de um projeto de retrabalho;
k) **utilizando os casos de teste para revisão de código:** cada módulo de código apresentará uma parte dos requisitos, por sua vez, os casos de testes são utilizados para validar se cada módulo oferece o esperado;
l) **verificação do código em relação aos casos de testes derivados de requisitos:** o último passo é o levantamento de casos de teste em relação aos casos de teste lógicos que foram arquitetados através da adição de dados e sua execução junto ao código, para comparar o resultado obtido ao resultado esperado.

> Após a execução de todos os casos de teste, pode-se assegurar que 100% dos casos de testes foram executados e o código pode ser liberado para produção.

Para os subcapítulos seguintes, este trabalho discutirá o processo de cobertura funcional dos testes de software.

3.1 COBERTURA FUNCIONAL

A descrição da cobertura funcional é necessária para assegurar a completude do software a ser testado, pois é o que garante que todas as funcionalidades do sistema estejam expostas em documentos e caso seja realizada alguma alteração ou melhoria no software consiga-se obter um conjunto de testes já criados para executar, certificando assim, que todas as entradas existentes no software tenham sido abrangidas.

A cobertura funcional foi definida por Silva (2007), como um método que é empregado para determinar a medida de quanto a funcionalidade do sistema está sendo avaliada. A cobertura funcional é um dos elementos relevantes na área de testes, pois possui o compromisso de assegurar a qualidade de software. Por intermédio dela referencia-se tudo o que está disponível para proceder em uma tela específica de um aplicativo que será testado.

Para ele, caso a cobertura funcional de um software não tenha sido feita de forma correta, ou seja, deixou-se de observar alguma funcionalidade do sistema, esta situação é chamada de buraco de cobertura. Cabe lembrar que caso seja identificado, o mesmo pode levar a uma avaliação incompleta do software, sendo assim é importante resolver este problema logo que for detectado.

Ao pôr em prática os testes funcionais, conforme afirma Silva (2007), uma das maiores dificuldades está em ter conhecimento de qual é o momento correto para cessar os testes assegurando que a avaliação esteja concluída, ou seja, certificar que todas as funcionalidades tenham sido analisadas. Para este caso o autor sugere que haja, pelo intermédio de algum procedimento, uma maneira de identificar se todas as funcionalidades efetivamente foram analisadas.

A cobertura funcional nada mais é que um método empregado para determinar a medida de quanto a funcionalidade do sistema está sendo avaliada e apresentar as funcionalidades do sistema que não foram abrangidas pelos testes. Uma cobertura completa favorece a excelência dos testes e, por conseguinte a melhoria do nível de qualidade do software. Além disso, torna-se mais transparente a percepção dos locais que não foram apropriadamente analisados e, consequentemente, deve-se concentrar na elaboração e no atingimento dos locais não avaliados, que são as áreas denominadas de buraco da cobertura. Estas lacunas devem ser percebidas e resolvidas sem demora, pois caso possua esse problema, automaticamente obstruirá a análise completa do software, dessa maneira possibilitará que falhas sejam toleradas, indevidamente, durante a etapa de testes (SILVA, 2007).

Um modelo descrito por Silva (2007) para determinar uma cobertura funcional completa é o *cross-product*, que também é denominado como *cross-coverage*. Esse modelo, segundo Grinwald et al (1998, tradução nossa) pode ser construído da seguinte maneira: inicia-se criando uma descrição semântica do modelo para descrever o tipo de eventos que serão cobertos, essa descrição contém um conjunto de atributos que tornam-se os atributos do modelo de cobertura *cross-product*. Para cada atributo, define-se o conjunto de todos os valores possíveis que o mesmo pode receber. Por fim, é definida uma lista de restrições que descrevem as combinações legais no *cross-product* dos valores do atributo. Logo, podemos considerar que, com base neste modelo, são executados testes em todos os cenários existentes dentro de um conjunto de funcionalidades.

No modelo de cobertura funcional *cross-product*, um exemplo pode ser exposto da seguinte maneira: na funcionalidade de acesso (autenticação) de um e-mail existe uma grande quantidade de testes a serem executados, tomando como base somente os campos de e-mail e senha, consegue-se determinar alguns casos de testes:

a) **caso de teste 1:** informar somente o e-mail;
b) **caso de teste 2:** informar somente a senha;
c) **caso de teste 3:** informar um e-mail válido com uma senha incorreta;
d) **caso de teste 4:** informar um e-mail inválido com a senha correta.

Nesta situação foram listadas somente quatro casos de testes a serem avaliados na funcionalidade do sistema. Para estes casos tratados, a princípio, considera-se como cobertura funcional, enquanto que para o restante dos casos de teste que não foram abrangidos denomina-se como buraco na cobertura.

Visto isto, o espaço de *cross-coverage* é construído com base em todas as funcionalidades de um software, avaliando as entradas que o usuário poderá realizar e testando todas elas, para certificar, assim, a completa cobertura funcional do sistema e, por conseguinte atestar a qualidade para o usuário final.

A criação de roteiros e casos de teste, além de garantir a cobertura de teste deve utilizar-se de parâmetros para avaliar se o teste não será realizado além do exigido.

3.2 DERIVAÇÃO FUNCIONAL

Casos de testes são aplicados dentro de roteiros de testes e possuem um passo-a-passo preciso de como o software será alimentado, além de definir de maneira exata o resultado esperado ao concluir a execução planejada.

Um caso de teste é conceituado por Fewster e Graham (1999, tradução nossa), como uma união de múltiplos testes que são efetuados sequencialmente em procura de uma finalidade: alcançar um domínio máximo acerca dos testes e diminuir a duração da aplicação dessa prática. Os autores ainda complementam que um caso de teste origina outros testes, que possuirão referências de valores que alimentarão o software, resultados esperados e outros dados auxiliares necessários para o teste, simplificando a realização desse procedimento.

O caso de teste é um elemento essencial para uma boa qualidade dos testes, segundo Myers (2004, tradução nossa), o caso de teste possui a incumbência de revelar falhas do software e quanto mais encontrar anomalias, consequentemente, seu êxito será maximizado.

Cartaxo (2006) afirma que, nos casos de teste são estabelecidas todas as funcionalidades existentes no produto, assegurando um aumento da

área de cobertura e, em vista disso, o testador poderá efetuar um a um, certificando que o software cumpre sua função equivalente ao que foi estipulado.

Após ser desenvolvido, o software é endereçado ao setor de teste, nesse momento é importante estar com os roteiros e casos de teste prontos, pois é com essas informações que saberá distinguir os requisitos funcionais e será capaz de testá-los. O testador deverá realizar a inserção das entradas de acordo com o que foi escrito e averiguar os resultados esperados de acordo com o que foi definido no caso de teste. Caso o resultado seja o previsto pode-se considerar que o comportamento está correto, caso contrário, esse item deverá ser anotado com o máximo de detalhes para ser enviado à equipe de desenvolvimento para corrigi-lo.

A derivação dos casos de teste é uma fase importante, pois para Jorgensen (1995, tradução nossa) é quando se determina que os casos de teste sejam constituídos por entradas e saídas. A entrada detém a circunstância preliminar e os estágios que serão seguidos para alcançar os resultados. Na saída é estabelecido o resultado previsto, ponderando a entrada inicial e por último o estado que o sistema pode apresentar. Estas são as maneiras para protocolizar um caso de teste completo, para que o testador possa fazer uso, com todas as informações necessárias e a derivação funcional mais adequada.

É importante possuir casos de teste que contemplem as derivações funcionais, isto é, que enumerem as funcionalidades do software com completude. Diante disso, compreende-se que com um caso de teste bem claro e escrito, aumentam-se as possibilidades de localizar a maioria dos defeitos presentes no software, além de minimizar o tempo, diminuir o custo e aumentar a produtividade.

É fundamental a elaboração dos casos de teste de forma manual, com o objetivo de alcançar os resultados esperados para equiparar com os resultados encontrados no sistema a ser testado. Porém, encontram-se técnicas de derivação de maneira automática, que simplificam esse processo, pois não haverá a necessidade de repeti-lo a cada melhoria ou correção. Para esse método, os casos de testes são elaborados automaticamente. A CONDADO é uma ferramenta com essas características e tem como objetivo

possibilitar o tratamento de aspectos de controle e dados de maneira única (SIQUEIRA, 2005).

Com os casos de teste elaborados, o testador precisa efetuá-los conforme foi descrito no roteiro e anotar cada resultado atingido, objetivando a descoberta de falhas. Com os resultados de cada execução confronta-se o resultado previsto em cada caso de teste e como resultado dessa comparação tem-se a decisão de indicar se o comportamento foi alcançado ou não. Enquanto existir erros sendo encontrados no sistema, o processo de solicitação da correção para a equipe de desenvolvimento será mantido e será necessário passar novamente pela avaliação da equipe de testes. Quando nenhuma falha for encontrada no processo de teste, o software estará pronto para ser distribuído para o cliente.

3.2.1 Técnicas de derivação funcional

A criação de roteiros e casos de teste sem a utilização de parâmetros, somente acarreta em testes realizados além do necessário, testes importantes que deixam de ser realizados – que influenciariam na localização de falhas - e tempo gasto sem necessidade.

Myers (2004, tradução), declara ser inviável a realização de testes exaustivos, pois considerando uma simples funcionalidade existe uma diversidade de situações a serem testadas e isso sobrecarregaria o processo de teste. Diante disso, utilizam-se métodos para levantamento dos roteiros e casos de teste. Para realizar essa seleção, os critérios mais pertinentes são:

a) **particionamento de equivalência:** segmenta as entradas de dados em classes que dispõem de características iguais, encontrando erros que eventualmente ocorrem em uma mesma categoria de dados. Esta técnica tem como objetivo diminuir a quantidade de defeitos isolados, já que os agrupa em um conjunto e, assim, o teste pode ser executado de um modo significativamente mais eficaz (PRESSMAN, 2006);
b) **análise do valor limite:** complementa o critério anterior, apesar disso as extremidades integradas às circunstâncias de entrada e saída são tratadas de uma forma mais precisa. Os testes são

selecionados nos limites das classes, considerando que é neste ponto que se centraliza o maior número de falhas. Resumidamente, este método diminui o total de roteiro e casos de testes, pois delimita os testes para os limites das classes (PRESSMAN, 2006);

c) **grafo causa-efeito:** os critérios anteriores possuem barreiras que restringem a elaboração de roteiros e casos de teste estabelecendo requisitos de teste fundamentados nas prováveis combinações das condições de entrada. Inicialmente, são avaliadas as prováveis condições de entrada (causas) e as ações do sistema (efeitos). Posteriormente, é criado um grafo confrontando as causas e efeitos avaliados e a partir disso, é elaborada uma tabela de decisão com o resultado da conversão do gráfico de causa-efeito. E por fim, os casos de teste são estruturados com base na conversão da tabela de decisão (PRESSMAN, 2006);

d) **matriz ortogonal:** esta técnica é adotada em circunstâncias em que o domínio da entrada é pequeno, mas grande demais para testes exaustivos. É um critério que auxilia na identificação de falhas associadas a uma categoria de falhas (PRESSMAN, 2006). Exemplo: considerando um software que tem quatro elementos de entrada: A, B, C e D, cada um desses elementos de entrada compreende três valores discretos relacionados a ele. Logo, existe 4^3 = 64 possíveis casos de teste. A escolha pela utilização desta técnica deve ponderar que é laborioso testar tudo, mas incorreto nenhum teste ser executado. Segundo Molinari (2008), para utilizar esta técnica, primeiramente, é exigido que seja definido a quantidade e quais combinações serão testadas. Em seguida, é delimitada a quantidade de valores que cada um dos elementos combinados obterá. Depois disso, são estabelecidas quantas e quais variáveis serão abordadas no teste. Como resultado dos dados levantados, é construída uma tabela onde o valor disponível para cada variável esteja presente no mínimo uma vez. Posteriormente, são criados os roteiros e casos de teste

com base na tabela elaborada anteriormente, certificando assim, que cada parâmetro de entrada terá um dos seus valores possíveis exposto a testes funcionais ao menos uma vez;

e) **experiência e conhecimento do especialista de teste:** é uma técnica não formal que é baseado na competência e destreza do especialista de teste (RIOS; MOREIRA, 2006), pois ele já possui uma familiaridade com as técnicas de testes e com as definições das diretrizes do negócio da empresa. Além disso, o especialista possui o conhecimento prático adquirido ao longo dos testes realizados anteriormente e das falhas encontradas. Com todo seu conhecimento o especialista de teste, consegue apontar os casos de teste, sem repetição, que deverão ser executados para manter a coerência com as funcionalidades que, historicamente, tendem a detectar falhas.

Uma peculiaridade significativa a todos os critérios citados anteriormente é a necessidade de que os requisitos funcionais estejam concluídos e coerentes ao construir os roteiros e casos de teste, de outro modo, serão considerados controversos os resultados dos testes (DELAMARO; MALDONADO; JINO, 2007).

4 CICLO DE VIDA DE TESTE

Na metodologia de teste de software precisa-se de estruturação para atingir o máximo de qualidade, buscando com que cada fase seja superada na sua plenitude e isso é estabelecido pelo ciclo de vida do teste. Para que esta técnica seja organizada deve-se segmentá-la em etapas com a finalidade de que todos os testes não sejam realizados somente em um momento, assim assegura-se a qualidade do software e o contentamento do usuário final.

A aptidão habitual em relação à definição do modelo de ciclo de vida de teste é de que seja utilizada de maneira nivelada ao modelo praticado no ciclo de vida do software, objetivando-se uma sincronia entre os modelos (SOFTEX, 2011).

As fases de testes, conforme Pressman (2002) são ponderadas como um método para o gerenciamento dos testes, com o objetivo de selecionar os testes e realizá-los por estágios, começando por intermédio de testes dedicados a uma categoria de componentes e finalizando quando os testes são empregados no sistema incorporado.

Pressman (2010) classificou a estruturação dos testes em três fases:

a) **teste de unidade:** também classificado como teste unitário ou teste de módulo, nesta fase são testadas as pequenas partes do sistema como as sub-rotinas, os métodos, classes e trechos menores de código, tem-se por finalidade a constatação de defeito em partes do sistema que não acarretam em falhas no sistema como um todo;
b) **teste de integração:** executa-se o teste quando o software apresenta-se na forma de um executável, objetiva-se nessa fase o teste da interface do sistema, bem como localizar defeitos originados da integração interna dos componentes do software;
c) **teste de sistema:** é nesta fase que o sistema é testado como um todo, confirmando que de fato a integração entre os módulos está correta, deve-se executar o software sob o panorama do cliente final, esmiuçando a finalidade do software em relação ao intuito original.

Além desses, existe o teste de regressão, que é uma técnica que possibilita mensurar o progresso da qualidade de software. Essa técnica é encarregada de realizar testes de software nas versões mais recentes do sistema, utilizando os mesmos testes realizados em versões anteriores, tendo como objetivo assegurar que não apareceram novas falhas em situações analisadas anteriormente.

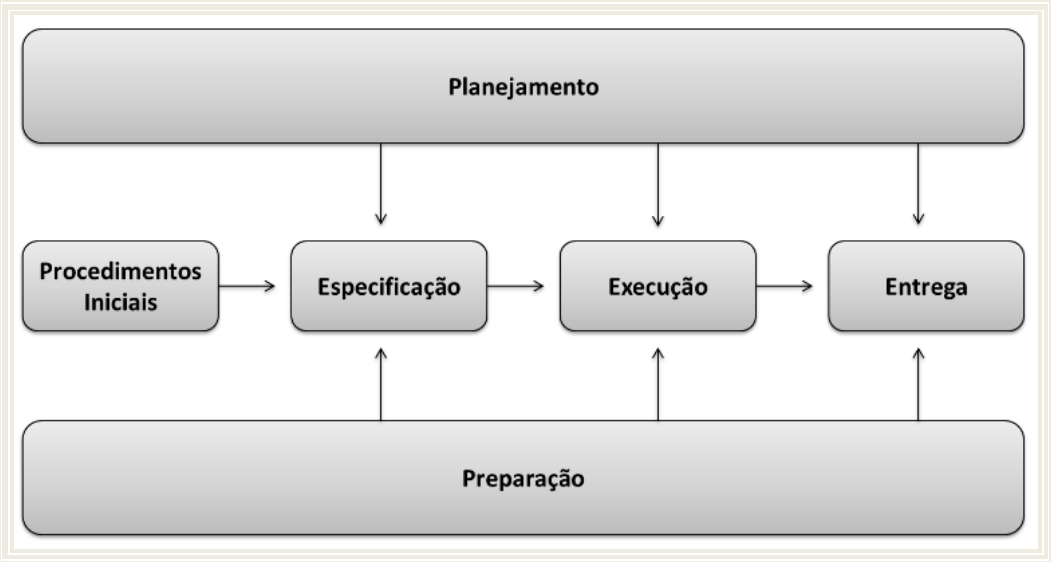
Além dessa estruturação feita por Pressman, o ciclo de vida de testes pode ser constituído por seis etapas, conforme Rios e Moreira (2006):

a) **procedimentos iniciais:** deve ser aplicada uma análise dos requisitos do negócio que originará o sistema de informação a ser testado, assegurando que esteja íntegro e sem ambiguidade. Conclui-se esta etapa com a criação do Guia Operacional de Teste – GOT;
b) **planejamento:** compreende no desenvolvimento e retificação da Estratégia e do Plano de Teste a ser empregado de forma a reduzir os principais contratempos do negócio e viabilizar as trajetórias para as próximas etapas. Essa atividade deve conservar-se até a finalização do projeto;
c) **preparação:** tem como objetivo essencial a organização do ambiente de teste (equipamentos, equipe, ferramentas de automação, hardware, rede e software), para que os testes sejam efetuados apropriadamente. Também deve manter-se até que o projeto seja concluído;
d) **especificação:** tem como finalidade a criação e adequação dos roteiros e casos de teste. Devem ser elaborados dinamicamente de forma simultânea ao transcorrer do projeto de teste;
e) **execução:** os testes são realizados em conformidade com os casos e roteiros de teste, fazendo uso dos mecanismos organizados na etapa de preparação. Caso seja utilizada alguma ferramenta de automação de testes, devem ser aplicados *scripts* de teste;
f) **entrega:** Nesta fase, o projeto de teste é concluído. Toda a documentação do projeto é finalizada/arquivada e serão mencionados todos os fatos ocorridos no projeto, que foram

avaliadas como imprescindíveis para o aperfeiçoamento do projeto.

A figura 5 mostra a estruturação apresentada acima de forma esquematizada. Percebe-se que as atividades de Planejamento e de Preparação conservam-se até a conclusão do projeto e ocorrem paralelamente aos procedimentos iniciais: especificação, execução e entrega, que ocorrem sequencialmente.

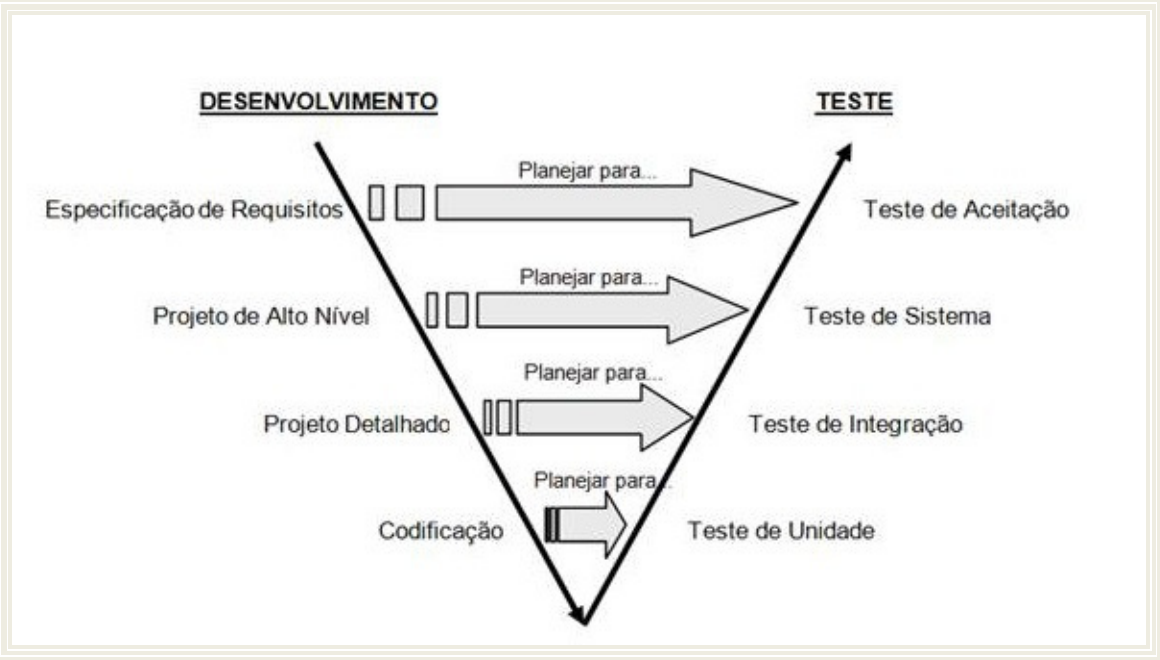
Figura 5 – Ciclo de vida de teste: modelo 3P x 3E



Fonte: Rios e Moreira (2006).

Outros modelos como V (figura 6), W e RUP são considerados ciclos de vida prontos, mas, isso não impede que a empresa que possui um produto a ser analisado crie um modelo que satisfaça as suas características. Para isso, será necessário o conhecimento de boas técnicas em teste de software, a percepção do que será priorizado, como se pretende alcançar os resultados esperados no teste e, não menos importante, o discernimento do momento adequado para introduzir o padrão elaborado (MOLINARI, 2008).

Figura 6 – Ciclo de vida de teste: Modelo em V



Fonte: Adaptado de Rios e Moreira (2006).

Independente do modelo escolhido para o ciclo de vida do teste de software é indispensável o aprofundamento do conhecimento em relação ao produto e a listagem de itens essenciais para o teste, uma vez que será necessário determinar qual modelo aparenta possuir menor risco, performance producente superior e evolução da excelência do produto. É preciso, da mesma forma, levar em conta que tão importante quanto essa alternativa é o emprego de análises ininterruptas no processo produtivo, com o objetivo de progredi-la quando for justificável em relação às recentes exigências ou defeitos encontrados.

O ciclo de vida do teste de software também possui componentes importantes para sua estruturação como as fases de teste.

4.1 FASE DE INTEGRAÇÃO

O teste de integração, de acordo com Inthurn (2001), é a fase do teste de software, na qual se verifica se os módulos testados individualmente também executam corretamente quando combinados em conjunto. Essa etapa ocorre posteriormente à fase de unidade, onde os módulos são testados separadamente.

Pressman (2002) assegura que essa fase tem como finalidade a descoberta de falhas relacionadas à interface. Nesse estágio é elaborado o fundamento do sistema, que foi definido no projeto. Ele ainda confirma que esse teste é desmembrado em dois tipos: incremental e não incremental ou *big bang*. São definidas por Pressman (2006):

a) **integração incremental:** as unidades são integradas gradativamente, desse modo é maximizada a expectativa de que seja completamente testado, ampliando a probabilidade de localizar falhas. Pode ser subdividida em: descendente (*top-down*), estratégia que inicia com a unidade principal e gradualmente integra as unidades dependentes a ela, e ascendente (*bottom-up*), técnica que começa integrando as unidades secundárias. Para Inthurn (2001), a integração incremental é a técnica mais conveniente para ser usada, sendo que é mais simples isolar as circunstâncias das falhas quando analisadas partes menores, em oposição a testar o sistema completo;
b) **integração não incremental ou *big bang*:** todos os componentes, programas, módulos ou subsistemas do software são combinados de uma única vez, ou seja, o teste é empregado sobre o sistema completo. Desta forma o esforço para preparação dos testes é minimizada, mas no sentido oposto, a complexidade para diagnosticar e reparar os defeitos é maior.

Independente da técnica escolhida nesta fase, ao final almeja-se possuir um produto integrado e preparado para ser exposto à próxima fase de testes (DELAMARO; MALDONADO; JINO, 2007).

4.2 FASE DE SISTEMA

A fase de teste de sistema tem como objetivo central colocar o produto de forma integral para validação, em outras palavras, após a conclusão de todas as fases anteriores, o produto é validado em sincronia com outros sistemas de software e componentes de hardware, que serão aproveitados

comumente após a sua liberação (INTHURN, 2001). Além disso, Delamaro, Maldonado e Jino (2007), afirmam que também devem ser explorados os requisitos não funcionais, que incorporam verificação de segurança, performance e robustez. E complementam que a maioria das organizações aplicam estratégias de constituir uma equipe independente para testar o sistema.

Essa averiguação da completude das funcionalidades é desempenhada com a execução dinâmica do sistema, avaliando se o comportamento das entradas válidas e inválidas corresponde à saída almejada (ROCHA; MALDONADO; WEBER, 2001).

Inthurn (2001) afirma que essa fase pode ser desmembrada em outros quatro testes:

a) **teste de recuperação:** é um teste de sistema que verifica a robustez de um software após submetê-lo a falhar de diversas maneiras. Objetiva-se com isso avaliar se o software consegue recuperar-se automaticamente, facilmente e completamente de uma falha (falhas no disco, falhas na interface, memória insuficiente, etc) ou haverá necessidade de intervenção humana, precisando nesse caso da avaliação do tempo de correção para definir se ele se localiza dentro da margem plausível;
b) **teste de segurança:** é um teste de sistema que tem como objetivo assegurar que o software estará protegido a eventuais tentativas ilegais de acesso, ou seja, os recursos de segurança elaborados no sistema devem blindá-lo deste acesso indevido;
c) **teste de estresse:** é um teste de sistema efetuado para confrontar os softwares com circunstâncias incomuns. Deste modo, o sistema é carregado com quantidades não usuais com o propósito de travá-lo. Com isso, pode-se supervisionar a perda de desempenho e sua suscetibilidade de apresentar erros no decorrer destas cargas. Para esta fase se faz necessário o emprego de quantidade, frequência e volumes incomuns, por exemplo, buscas demasiadas de dados em disco, acréscimo elevado de índices de dados, abertura simultânea de muitas janelas, sempre com a finalidade de encontrar falhas de memória,

uso de arquivos com formato diferente dos desejados pelo software;

d) **teste de desempenho:** é um teste de sistema realizado com a finalidade de encontrar o limite de processamento de dados e analisar a eficácia do tempo de resposta em cenários e configurações pré-estabelecidos. É importante salientar que este teste, comumente, é praticado simultaneamente ao teste de estresse.

4.3 FASE DE REGRESSÃO

Esta fase tem o intuito de assegurar que o software conserve-se íntegro depois de novos testes efetuados, ou seja, essa etapa do teste acontece após o produto já ter sido testado e voltar para correções, pois ao regressar para o setor de testes, o software não pode possuir falhas em funcionalidades, anteriormente, já analisadas e validadas através dos testes (RIOS; MOREIRA, 2006).

O teste de regressão é definido por Sommerville (2003) como uma etapa que certifica que todos os testes, executados anteriormente no software, permaneçam proporcionando a conservação da integridade do produto ao passar por alguma modificação.

Os testes de regressão são imprescindíveis para garantir a continuidade do desempenho do sistema (BASTOS et al, 2007; DELAMARO;MALDONADO;JINO, 2007; PRESSMAN, 2006), visto que as alterações constantes (correções, melhorias ou novas funcionalidades) imutavelmente acometem o software pelo volume de falhas introduzidas ao produto (SOMMERVILLE, 2003). Este é um elemento que consolida o teste de regressão essencial para o processo de testes em softwares que são adequados constantemente para buscar o aperfeiçoamento da qualidade do produto. Para estes casos, citamos os softwares com requisitos funcionais definidos por legislações, como os sistemas de contabilidade, que estão sempre sofrendo modificações, principalmente em relação à criação e à alteração de impostos.

5 MATRIZ DE RASTREABILIDADE

A matriz de rastreabilidade, também conhecida como Traceability Rastreability Matrix (TRM), se refere a um documento de apoio que auxilia a visualização dos relacionamentos entre requisitos e outros artefatos ou objetos. Com a rastreabilidade entre requisitos e casos de testes possibilita-se, entre outras coisas, a visualização da quantidade de casos de teste que foram destinados para um determinado requisito como também permite verificar se algum requisito não conteve casos de teste designados.

A cada ciclo de teste são levantados dados históricos dos casos de testes que foram criados com base em uma técnica de derivação funcional. Essas informações são armazenadas por completo em TRM's, que por sua vez são vinculadas a um determinado ciclo de teste. Na figura 7 é apresentada a estrutura de uma matriz de rastreabilidade.

Figura 7 – Estrutura de uma TRM de casos de teste

Código Caso de teste	Descrição do caso de teste	Planejado	Executado	Passou	Falhou	Issue (Cód. Erro)	Quem testou	Quando	Tempo executado (min.)
TC 1.0	Descrição 1								
TC 1.1	Descrição 2	X	X	X					
TC 1.2	Descrição 3								
TC 1.3	Descrição 4	X	X	X					
TC 1.4	Descrição 5								
TC 1.5	Descrição 6	X	X		X	1010	Testador 1	25/11/2014	10
TC 1.6	Descrição 7	X							
TC 1.7	Descrição 8	X	X	X					
TC *n*	Descrição *n*								

Fonte: Do autor.

A matriz de rastreabilidade armazena as seguintes informações:

a) **código do caso de teste:** serve para identificá-lo;
b) **descrição do caso de teste:** apresenta todas as informações necessárias (passo a passo) para sua correta execução;
c) **planejado:** indica se a execução do caso de teste foi planejada.
d) **executado:** informa se o caso de teste foi executado;
e) **passou:** indica, após a execução do passo a passo, se o caso de teste possui o comportamento esperado que foi descrito;
f) **falhou:** confirma, após a execução do passo a passo, se o caso de teste possui o comportamento diferente ao que foi anotado em sua descrição. Para este caso será necessário o preenchimento do Issue;

g) **issue (Cód. Erro):** informa o código do problema e todas as informações necessárias;

h) **quem testou:** indica o testador que executou o caso de teste;

i) **quando:** apresenta a data em que o caso de teste foi executado;

j) **tempo executado (min.):** indica o tempo gasto na execução do caso de teste.

A rastreabilidade de software tem sido reconhecida como um fator importante para qualquer fase de desenvolvimento de um sistema de software e processo de manutenção (SHERBA; ANDERSON, 2003), que contribui para a qualidade do produto final (SPANOUDAKIS; ZISMAN,2005), apesar disso, o suporte atual à rastreabilidade é inadequado.

A utilização da TRM neste trabalho será feita com base no Princípio de Pareto, concebido por Joseph M. Juran, que indica que para muitos fenômenos 80% das consequências são decorrentes de 20% das causas, princípio também conhecido como princípio 80-20 e com base na matriz de priorização GUT.

A partir dessa teoria, para os novos ciclos de testes serão priorizados os casos de testes referentes aos pontos do sistema que possuem histórico de apresentação de erros (que são as causas) para então localizar as falhas (que são as consequências). Como consequência disso, o tempo destinado aos testes será priorizado nos locais onde existe a tendência deles ocorrerem, deixando os casos de teste com menor propensão para serem executados por último, caso sobrar tempo no cronograma. Com a utilização desse documento (TRM), espera-se manter ou aumentar a garantia em relação à funcionalidade, pois o foco será destinado para o local onde possivelmente ocorrerão a maioria das falhas, utilizando um tempo menor para realizar este procedimento.

5.1 GRAVIDADE, URGÊNCIA E TENDÊNCIA (GUT)

A matriz de priorização GUT é uma ferramenta utilizada com a finalidade de priorizar as adversidades e dessa forma corrigi-las, com base nas suas Gravidades, Urgências e Tolerâncias (BEHR; MORO; ESTABEL, 2008).

Os autores salientam que a GUT pode ser utilizada para análise de qualquer matéria e para classificação de diversos assuntos. Além disso, por se tratar de uma tomada de decisão, torna-se ainda mais autêntica a elaboração dessa metodologia em grupo. De acordo com eles, a matriz GUT fundamenta-se na interpretação da Gravidade, Urgência e a Tendência das adversidades enfrentadas, sendo:

a) **gravidade:** evidencia o efeito da adversidade explorada, caso ela venha a ocorrer. Neste caso, são examinadas algumas questões como: tarefas, pessoas, resultados, metodologias, organizações, etc. Observa-se constantemente seu impacto a médio e longo prazo, caso a adversidade em questão não esteja solucionada;

b) **urgência:** demonstra o prazo, o tempo acessível ou indispensável para a elucidação de uma especificada adversidade averiguada. Quanto maior a urgência, menor será o tempo acessível. É sugerido que seja realizado o seguinte questionamento: "A elucidação desta adversidade pode postergar ou deve ser priorizada?";

c) **tendência:** destaca a capacidade de desenvolvimento da adversidade e a possibilidade desta transformar-se em algo maior com o passar do tempo. É uma estimativa da tendência de aumento, diminuição ou desaparecimento da adversidade. Sugere-se que seja realizado o seguinte questionamento: "Se a adversidade não for elucidada neste momento, ela tende agravar aos poucos ou agravará abruptamente?".

Cada adversidade deve ser pontuada de 1 a 5 em cada parâmetro, devendo utilizar como regra, para a pontuação, a figura 8.

Figura 8 – Matriz de priorização GUT

Nota	Gravidade	Urgência	Tendência
5	Extremamente grave	Extremamente urgente	Se não for resolvido, piora imediatamente
4	Muito grave	Muito urgente	Vai piorar em curto prazo
3	Grave	Urgente	Vai piorar em médio prazo
2	Pouco grave	Pouco urgente	Vai piorar em longo prazo
1	Sem gravidade	Sem urgência	Sem tendência de piorar

Fonte: Behr, Moro e Estabel (2008).

Posteriormente à pontuação dos parâmetros para cada um dos casos de teste, deve-se multiplicar na horizontal os pontos de cada adversidade, gerando um resultado chamado “Grau crítico” (Gravidade * Urgência * Tendência), pelo qual as adversidades são classificadas.

Para uma melhor avaliação das prioridades, pode-se utilizar o princípio de Pareto conjuntamente a esta ferramenta.

5.2 PRINCÍPIO DE PARETO

O Princípio de Pareto, criado pelo economista italiano Vilfredo Pareto, aproximadamente no ano de 1897, é uma estrutura elaborada com embasamento na análise sobre o desequilíbrio na distribuição de renda e riquezas da Itália. Nesta análise, Pareto comprovou que a maior parte da renda total centralizava-se em uma menor parcela da população, sendo que 20% da população detinha 80% da riqueza, na mesma proporção que o restante da população, 80% detinha apenas 20%. Essa estrutura foi denominada como Princípio 80/20 (MARSHALL, 2008).

Segundo Koch (2006), esse princípio tem como fundamento a lei do mínimo esforço, que busca concentrar e não depositar esforços onde se atinge baixos resultados. Em 1949, o professor de filosofia da Universidade de

Harvard George K. Zipf pesquisou a fundo a lei do mínimo esforço, na qual as pessoas visavam reduzir seus trabalhos observando o fundamento de que 20% de qualquer recurso são equivalentes a 80% da atividade obtida através desse recurso. Zipf possuía mais mecanismos do que Pareto para fundamentar suas análises e tinha a sua disposição estatísticas populacionais, livros de filosofia e leis de comportamentos industriais que acarretaram em uma melhor compreensão do princípio 80/20 e consequentemente ele reconstituiu o princípio de Pareto.

O 80/20 é um simples simbolismo de uma vinculação muito desigual entre causas e efeitos que precisamente são uma tendência onde 20% das causas ocasionam 80% dos resultados almejados.

Koch (2006) assinalou alguns modelos relevantes, do princípio de Pareto, em seu livro:

a) cinco pessoas se reúnem para jogar pôquer. Presume-se que uma dessas pessoas 20% vença no mínimo 80% das apostas;
b) menos de 20% das celebridades conquistam mais de 80% da mídia, ao passo que mais de 80% dos livros comercializados são de 20% dos autores;
c) mais de 80% das riquezas minerais são produzidas por 20% da superfície terrestre;
d) menos de 20% das guerras acarretam 80% das vítimas.

Podem ser citados anda alguns itens que beneficiam na aplicação do Princípio 80/20, como:

a) facilitar o entendimento da abrangência dos problemas, possibilitando salientar a cautela nos campos mais impactantes;
b) é mais simples minimizar a ocorrência do problema mais constante ao invés de minimizar uma ocorrência incomum;
c) soluções ágeis na elucidação de problemas, gerando segurança na equipe;
d) algumas circunstâncias originam diversos problemas; a correção de uma circunstância de um problema geralmente minimiza ou reduz o acontecimento de outros problemas.

O princípio 80/20 em conjunto com a matriz de priorização GUT interfere diretamente no planejamento dos testes bem como na definição dos planos e casos de teste.

6 PLANEJAMENTO DE TESTES

Planejamento de teste envolve programação e estimativa do processo de teste, estabelecendo padrões de processo e descrevendo os testes que devem ser realizados. (SOMMERVILLE, 2001, tradução nossa).

Planejamentos de teste são destinados a engenheiros de software envolvidos na concepção e realização de testes do sistema. Eles ajudam a equipe técnica obter uma imagem geral dos testes de sistema e colocam o seu próprio trabalho neste contexto (SOMMERVILLE, 2001, tradução nossa).

O autor ainda afirma que o planejamento do teste é particularmente importante no desenvolvimento de grandes sistemas de software. Assim como, estabelecem os procedimentos de teste, o plano de teste define os recursos de hardware e software que são necessários. Isto é útil para os gestores do sistema, que são responsáveis por garantir que estes recursos estejam disponíveis para a equipe de teste. Planos de teste devem normalmente incluir quantidades significativas de contingência para que derrapagens na concepção e implementação possam ser acomodados e os funcionários destacados para outras atividades.

Ele ainda declara que planos de teste não são documentos estáticos, mas que estão em constante evolução durante o processo de desenvolvimento. Atrasos em outras fases do processo de desenvolvimento também podem modificá-los, pois se uma parte de um sistema está incompleta, o sistema como um todo, não pode ser testado. O plano de teste, então, deve ser revisto e os testadores direcionados para alguma outra atividade até o software estar novamente disponível.

6.1 DEFINIÇÃO DO PLANO DE TESTE

O plano de teste é um documento empregado para administrar o projeto de teste. Segundo Rios (2007), esse documento deve ser estruturado com: identificador do plano de teste, introdução, itens de teste, funcionalidades a serem testadas, funcionalidades a não serem testadas, abordagem do teste, critérios de liberação/falha dos itens, requisitos de suspensão e retomada,

entregas do teste, tarefas do teste, necessidades de ambientes, responsabilidades, necessidades de equipe e de treinamento, cronograma, riscos e aprovações. O autor ainda afirma que este plano, que é definido pela norma IEEE 829[2], deve ser elaborado pelo gerente de testes, pois é ele quem prepara a estratégia de teste para sua equipe.

6.2 DEFINIÇÃO DOS CASOS DE TESTE

O caso de teste define um elemento de teste que será efetuado pelo testador e caracteriza uma circunstância específica a ser testada e é constituído por valores de entrada, restrições para a sua realização e um resultado esperado (CRAIG; JASKIEL, 2002). Este documento deve ser arquitetado com: identificador, itens de teste, especificações de entrada e de saída, necessidades de ambiente, requisitos ou procedimentos especiais e dependências entre casos de teste (RIOS, 2007). De acordo com Myers (2004, tradução nossa), o caso de teste deve indicar com precisão a saída esperada e os comportamentos esperados do procedimento. O autor ainda reitera que na elaboração dos casos de teste pretende-se, em uma condição adequada, localizar um subconjunto dos casos de teste que possua maior probabilidade de localizar a maioria das falhas.

[2] Disponível em <http://dis.unal.edu.co/~icasta/GGP/_Ver_2013_1/Documentos/Normas/829-2008.pdf>. Acesso em 09 nov. 2014.

7 TRABALHOS CORRELATOS

Neste capítulo serão listados alguns trabalhos que possuem ligação direta ou indireta a este trabalho que está sendo desenvolvido.

7.1 DESENVOLVIMENTO DE UMA METODOLOGIA BASEADA NA AUTOMAÇÃO DO PROCESSO DE TESTES DE SOFTWARE COMO ESTRATÉGIA PARA A GARANTIA DA COBERTURA FUNCIONAL

Este trabalho de conclusão de curso foi proposto pela acadêmica Camilla Damiani do curso de Ciência da Computação, pela Universidade do Extremo Sul Catarinense – UNESC, no ano de 2012, a fim de obter o título de Bacharel, sob orientação do Prof. MSc. Gustavo Bisognin.

A proposta do trabalho foi estabelecer padrões para cobertura de testes funcionais de forma automática, proporcionando a redução de defeitos causados por falhas no processo de teste. Além disso, utilizou a matriz de rastreabilidade para facilitar a visualização dos relacionamentos entre requisitos e outros artefatos ou objetos.

7.2 A IMPORTÂNCIA DO PROCESSO DE TESTE PARA A QUALIDADE DO SOFTWARE

Esta monografia foi proposta pela acadêmica Tilene Corsi Cavalcanti do curso de Tecnologia em Informática para Gestão de Negócios, da Faculdade de Tecnologia da Zona Leste – FATECZL, no ano de 2009, a fim de obter o título de Tecnóloga em Informática para Gestão de Negócios, sob orientação do Prof. Msc. Antônio Tadeu Pellison.

A proposta do trabalho foi comprovar que a tecnologia está encaixada no mundo corporativo. Foi indicado que a atividade de teste de software é um método fundamental para garantia da qualidade do produto. Nesse trabalho foram realizados estudos de casos em uma corporação e comparativos para determinar que a execução de testes de software é essencial para qualidade do sistema disponibilizado ao usuário.

7.3 UTILIZAÇÃO DE TESTES AUTOMATIZADOS PARA AMPLIAR A COBERTURA FUNCIONAL NA MANUTENÇÃO DE SOFTWARE

Este trabalho de conclusão de curso foi proposto pelo acadêmico Cleverson Reinert do curso de Ciência da Computação, pela Universidade do Extremo Sul Catarinense – UNESC, no ano de 2012, a fim de obter o título de Bacharel, sob orientação do Prof. MSc. Gustavo Bisognin.

A proposta do trabalho foi aplicar testes automatizados na cobertura das funcionalidades do software, empregando critérios formais ou não formais para derivação dos casos de testes, como por exemplo, teste de matriz ortogonal, particionamento de equivalências ou o próprio conhecimento e experiência do especialista de testes. Objetivando reduzir o número de defeitos e o tempo total de execução de testes, garantindo maior qualidade e redução dos custos associados ao projeto de desenvolvimento de software.

7.4 UMA METODOLOGIA PARA TESTE DE SOFTWARE NO CONTEXTO DA MELHORIA DE PROCESSO

Este artigo foi pesquisado por Adalberto Nobiato Crespo, Odair Jacinto da Silva, Carlos Alberto Borges, Clênio Figueiredo Salviano, Miguel de Teive, Argollo Junior e Mario Jino, no ano de 2012. Esse trabalho teve contribuição de três instituições de Campinas-SP: Ampla Consultoria em Informação, Centro de Pesquisas Renato Archer (CenPRA) e UNICAMP e de uma instituição da cidade de Itatiba-SP: a Universidade de São Francisco.

A proposta do trabalho foi elaborar uma metodologia para inserção ou aperfeiçoamento da técnica de teste em empresas desenvolvedoras de software. Este artigo evidencia considerações sobre teste no âmbito da qualidade de software, uma visão geral e resumida da metodologia desenvolvida, uma aplicação prática desta metodologia e conclusões.

8 CONSTRUÇÃO DE UM PROCESSO DE PRIORIZAÇÃO DE CASOS DE TESTES

O emprego de testes nos softwares é indispensável para a garantia da qualidade do mesmo, principalmente para softwares que passam por correções e aperfeiçoamentos contínuos, que segundo Sommerville (2003), possuem uma tendência maior para apresentar falhas.

Esta situação torna indispensável uma análise com base na priorização dos casos de testes referentes aos pontos do sistema que historicamente revelam a presença de erros. Fundamentado nisso, o tempo reservado aos testes será priorizado nos pontos em que existe a predisposição deles ocorrerem, postergando os casos de teste com menor propensão para serem executados por último. Essa técnica visa conservar ou ampliar a garantia em relação à funcionalidade, pois o foco será designado para o local em que teoricamente ocorrerá a maioria das falhas, utilizando um tempo menor para a efetuação deste procedimento.

O processo de priorização de casos de teste foi realizado com base na matriz de rastreabilidade, sendo que a escolha dos casos de teste se sucedeu por intermédio de método de derivação não formal, fundamentado no conhecimento e experiência do especialista de teste.

O presente trabalho é a construção de uma técnica para priorizar os casos de teste na validação das funcionalidades do sistema, utilizando como apoio a matriz de rastreabilidade. Desta forma, objetiva-se a redução do tempo total gasto para executar os casos de teste, porém garantindo a qualidade do software, visto que serão priorizados os casos de teste que historicamente apresentam erros. Para isso faz-se necessário a elaboração de uma metodologia com etapas bem divididas.

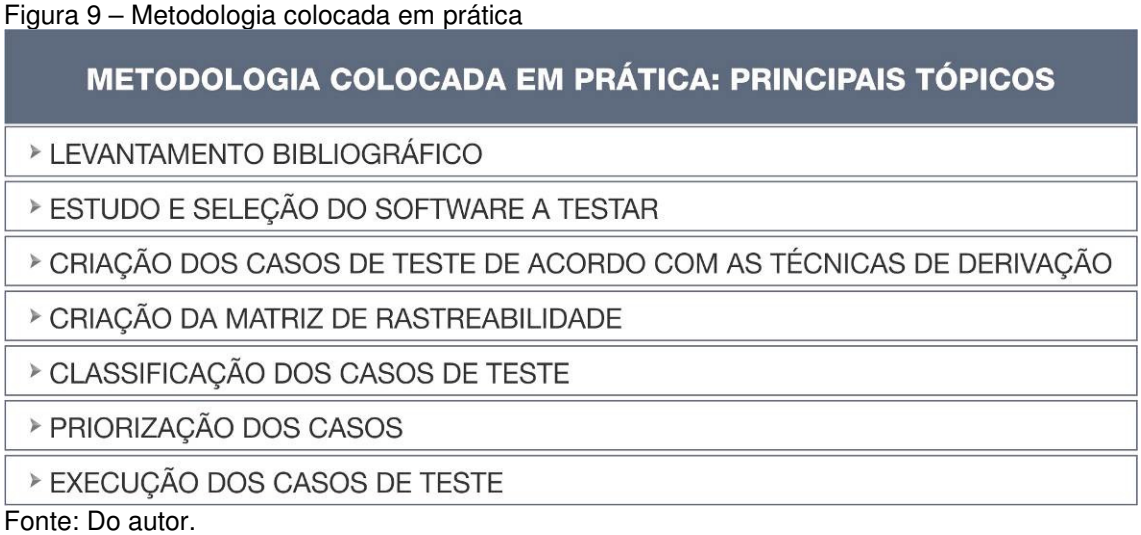
8.1 METODOLOGIA

A metodologia colocada em prática para a realização do trabalho foi subdividida em oito estágios, abrangendo desde o levantamento bibliográfico, passando pela análise dessas informações, pela pesquisa em relação a criação

dos casos de teste de acordo com as técnicas estudadas, a montagem da matriz de rastreabilidade a partir dos casos de testes criados, finalizando com a execução de métodos para classificar e priorizar casos de teste que serão executados no software escolhido.

O presente trabalho foi produzido conforme demonstra a figura 9, que expõe de forma esquematizada as fases essenciais para o desenvolvimento do trabalho, que serão abordadas nos subcapítulos subsequentes.

Figura 9 – Metodologia colocada em prática



Fonte: Do autor.

8.1.1 Levantamento bibliográfico

Para a elaboração e desenvolvimento do trabalho foi imprescindível a realização de pesquisas em diversos meios como a biblioteca da UNESC, em livros e artigos disponibilizados gratuitamente na internet.

Também foram realizadas pesquisas nas bases de dados disponibilizadas através da funcionalidade "Biblioteca Virtual" existente no site da UNESC.

Nas pesquisas realizadas constatou-se que as fontes acessadas eram confiáveis e possuíam relevância científica.

Percebeu-se uma grande quantidade de trabalhos direcionados para a área de engenharia de software, especialmente para o foco deste trabalho, que é o Teste de Software, tanto na forma de artigos, como em dissertações de graduação e teses de mestrado e doutorado.

No desdobramento desta pesquisa verificou-se uma extensa quantidade de publicações em língua inglesa, traduzidas com aprimoramento da gramática e da semântica, aplicando para isso conhecimentos em língua inglesa, assim como de termos técnicos do ramo da computação.

É importante destacar ainda que essa pesquisa bibliográfica proporcionou além da colaboração para o desenvolvimento do trabalho, um relevante ganho profissional, através dos conhecimentos obtidos.

Após a elaboração da fundamentação teórica, foi pesquisado e escolhido um software público para ser utilizado no trabalho.

8.1.2 Estudo e seleção do software para realizar a execução dos casos de teste

O software desktop escolhido para a execução dos casos de teste foi o IRPF2015 - Declaração de Ajuste Anual, Final de Espólio e Saída Definitiva do País da Receita Federal. Foram analisadas as telas de Identificação do Contribuinte (figura 10) e do Cadastro de Dependentes (figura 11).

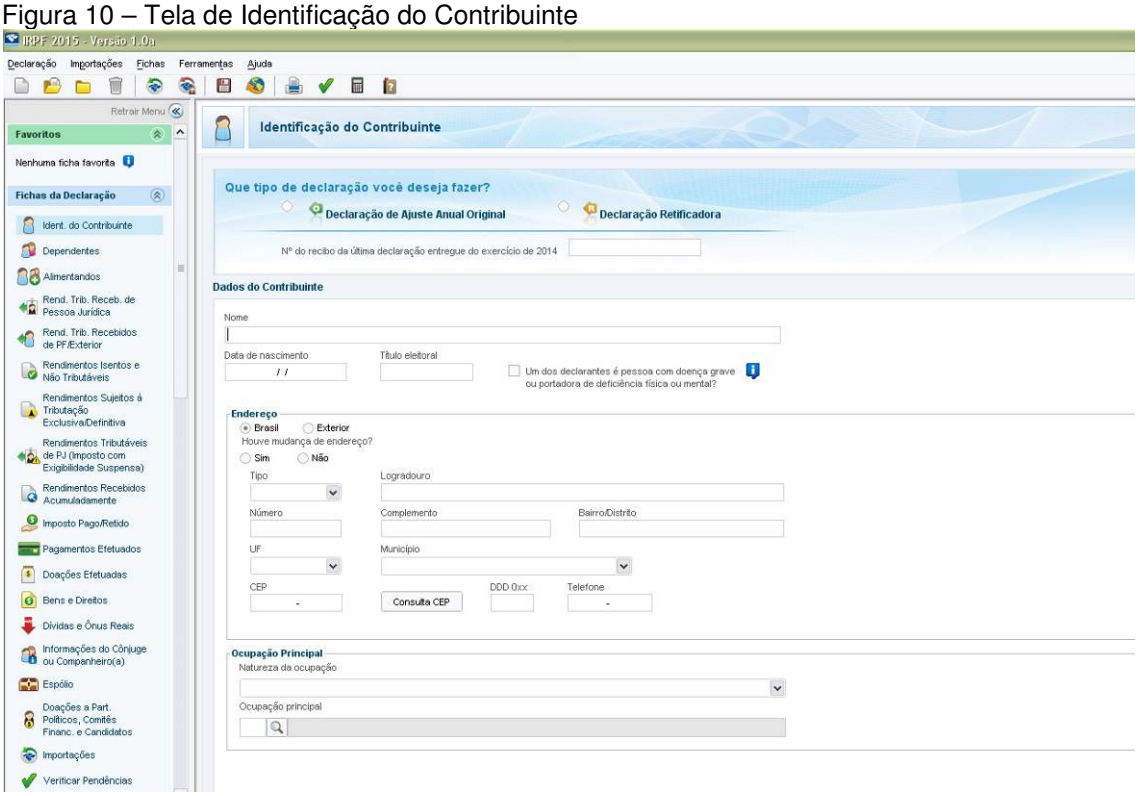
O Imposto de Renda de Pessoa Física, segundo a Receita Federal é um imposto federal que acomete todos os cidadãos que receberam rendimentos tributáveis acima de um valor estabelecido. Anualmente é exigido a esse contribuinte o fornecimento de informações através da Declaração de Ajuste Anual - DIRPJ, para que seja averiguado se o contribuinte terá débitos (pagamento de imposto) ou créditos (devolução de imposto). O imposto é determinado a partir da renda do contribuinte. A alíquota oscila equivalentemente de acordo com a renda tributável. Caso a pessoa não possua renda maior que o valor estabelecido, será considerada isenta.

Este software oferece diversos benefícios aos cidadãos que usufruem desse sistema, como a facilidade para declarar, eliminação de despesas com impressão e consulta online, em tempo real, da restituição, principalmente se comparado aos anos anteriores quando a declaração era um formulário preenchido manualmente ou preenchido eletronicamente e entregue em um disquete, além disso, não era possível consultar a situação da

declaração, visto que era necessário aguardar notificação da Receita Federal em casa.

Escolheu-se esse software, por ser um sistema que é disponibilizado a todos os contribuintes e sua liberação coincidiu com o início do desenvolvimento do trabalho proposto. É primordial que este software não apresente falha, para isso, torna-se importante a realização de testes utilizando critério de priorização das principais funcionalidades do sistema, sobretudo as que historicamente apresentam defeitos.

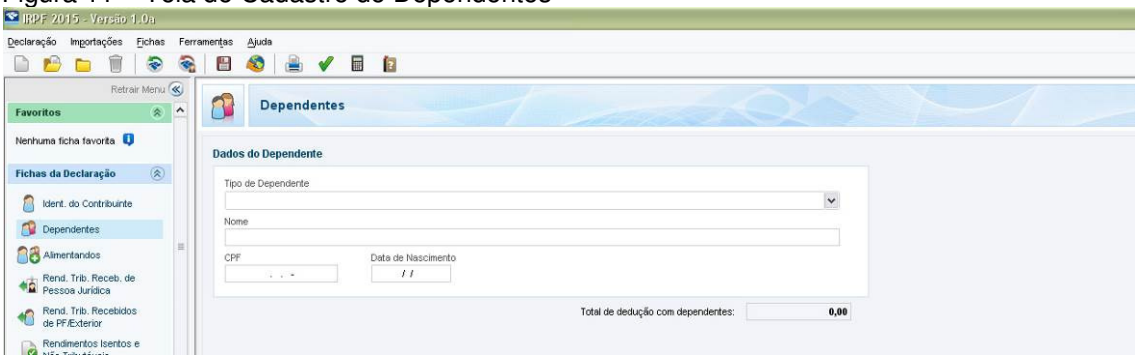
Figura 10 – Tela de Identificação do Contribuinte



Fonte: Do autor.

A figura 10 apresenta parte da tela de Identificação do Contribuinte, onde são preenchidos os dados pessoais, endereços, contato e informações profissionais.

Figura 11 – Tela de Cadastro de Dependentes



Fonte: Do autor.

A figura 11 apresenta parte da tela do Cadastro de Dependentes, onde são preenchidos os dados pessoais dos dependentes do contribuinte.

Com a definição do software a ser utilizado no trabalho proposto, será necessário o aprofundamento dos conceitos das técnicas de derivação funcional, para proceder com o levantamento dos casos de teste.

8.1.3 Criação dos casos de teste de acordo com as três técnicas de derivação mais utilizadas

Os estudos destacaram que os métodos de derivação fundamentados em particionamento de equivalências, análise do valor limite, grafo causa-efeito, matriz ortogonal e experiência e conhecimento do especialista de teste podem ser aplicados de maneira suplementar. Para isso, torna-se essencial a aplicação de dois ou mais métodos de derivação conjuntamente para proporcionar casos de teste que alcancem maior produtividade para obter maior qualidade no software.

Foram escolhidos três dos métodos de derivação existentes - particionamento de equivalências, análise do valor limite e experiência e conhecimento do especialista de teste - para a criação dos casos de teste com base em seus conceitos.

É relevante considerar que conforme Rios e Moreira (2006), o método fundamentado na experiência e conhecimento do especialista de teste, pode ser desenvolvido com a pretensão de alcançar a máxima qualidade dos testes e do software.

Diante disso e por ser o método empregado na empresa em que o pesquisador trabalha foram escolhidos os casos de teste criados a partir dos conceitos da derivação fundamentada na experiência e conhecimento do especialista de teste, tendo como objetivo alcançar resultados mais efetivos. Na figura 12 são demonstrados os casos de testes criados para a validação do campo CPF da tela de Identificação do Contribuinte.

Figura 12 – Casos de teste levantados com base na derivação fundamentada na experiência e conhecimento do especialista de teste

Casos de teste (Criar nova declaração)	
CPF	**Saída esperada**
377.845.609-15	Aceitar CPF
377.845.609-16	Erro: Número de inscrição no CPF inválido
999.999.999-99	Erro: Número de inscrição no CPF inválido
[sem preenchimento]	Erro: O campo indicativo "Número de inscrição no CPF" não foi informado

Fonte: Do autor.

A partir da criação dos casos de teste será realizada a montagem da matriz de rastreabilidade para cada ciclo de teste.

8.1.4 Criação da matriz de rastreabilidade

A matriz de rastreabilidade (TRM) armazena de forma íntegra as informações, levantadas dos dados históricos, dos casos de teste. Esse armazenamento de outro modo é relacionado a um ciclo de teste estabelecido.

Utilizando o software Microsoft Excel 2010 foram montadas sete tabelas, sendo uma planilha principal que contém a tabela dos ciclos de testes realizados e outras seis planilhas contendo as TRMs de casos de teste.

O requisito funcional escolhido para elaborar cada ciclo foi o sistema operacional utilizado na estação de trabalho. Este primeiro documento da matriz de rastreabilidade possui um link para acesso direto às planilhas de TRM de casos de teste, como apresentado na coluna Ciclo da figura 13. Baseado nessa planilha será possível averiguar os casos de testes planejados e executados.

Figura 13 – Tabela dos ciclos de teste

Ciclo	SO	Funcionalidade
1	Windows 8	Identificação do contribuinte
2	Windows 7	Identificação do contribuinte
3	Windows XP	Identificação do contribuinte
4	Windows 8	Dependentes
5	Windows 7	Dependentes
6	Windows XP	Dependentes

Fonte: Do autor.

Na figura 14 é apresentada a TRM 1.0, adotada para evidenciar os casos de teste planejados e executados no preenchimento das telas de Identificação do contribuinte e Cadastro de Dependentes do IRPF.

Figura 14 – TRM de casos de teste

Código Caso de teste	Descrição do caso de teste	Planejado	Executado	Passou	Falhou	Issue (Cód. Erro)	Quem testou	Quando	Tempo executado (min.)
TC 1	Campo CPF - 377.845.609-15								
TC 2	Campo CPF - 377.845.609-16								
TC 3	Campo CPF - 999.999.999-99								
TC 4	Campo CPF - [Sem preenchimento]								
TC 5	Campo Nome - Romário Maurício								
TC 6	Campo Nome - Romário Maurício 2								
TC 7	Campo Nome - Romário Maurício !								
TC 8	Campo Nome - [Sem preenchimento]								
TC 9	Campo Tipo de declaração - Ajuste Anual								
TC 10	Campo Tipo de declaração - Retificadora								
TC 11	Campo Tipo de declaração - [Sem escolha da opção]								
TC 12	Campo Nº último recibo - 128471140682								
TC 13	Campo Nº último recibo - 111111111111								
TC 14	Campo Nº último recibo - [Sem preenchimento]								
TC 15	Campo Data Nascimento - 29/02/2000								
TC 16	Campo Data Nascimento - 31/04/2000								
TC 17	Campo Data Nascimento - 29/02/2001								
TC 18	Campo Data Nascimento - [Sem preenchimento]								
TC 19	Campo Título eleitoral - 4356870906								
TC 20	Campo Título eleitoral - 4356870907								
TC 21	Campo Título eleitoral - [Sem preenchimento]								
TC 22	Campo Tipo de Logradouro - Rua								
TC 23	Campo Tipo de Logradouro - [Sem escolha da opção]								
TC 24	Campo Logradouro - Acre								
TC 25	Campo Logradouro - [Sem preenchimento]								
TC 26	Campo Número - 2								
TC 27	Campo Número - [Sem preenchimento]								
TC 28	Campo Complemento - Apartamento, bloco 2								
TC 29	Campo Complemento - [Sem preenchimento]								
TC 30	Campo Bairro - Próspera								
TC 31	Campo Bairro - [Sem preenchimento]								
TC 32	Campo UF - SC								
TC 33	Campo UF - [Sem escolha da opção]								
TC 34	Campo Município - Criciúma								
TC 35	Campo Município - [Sem preenchimento]								
TC 36	Campo CEP - 88813-210								
TC 37	Campo CEP - [Sem preenchimento]								
TC 38	Campo DDD - 48								
TC 39	Campo DDD - [Sem preenchimento]								
TC 40	Campo Telefone - 3461-2000								
TC 41	Campo Telefone - [Sem preenchimento]								
TC 42	Campo Natureza da ocupação - Militar								
TC 43	Campo Natureza da ocupação - [Sem escolha da opção]								
TC 44	Campo Ocupação principal - Militar da aeronáutica								
TC 45	Campo Ocupação principal - [Sem escolha da opção]								

Fonte: Do autor.

Com a criação da matriz de rastreabilidade será imprescindível a realização de uma classificação para determinar o grau crítico de cada caso de teste, resultando assim na utilização da matriz de priorização GUT.

8.1.5 Classificação dos casos de teste com a matriz de priorização GUT

A priorização torna-se vital para inteirar-se por qual ponto começar o teste, servindo como uma classificação, de modo que primeiramente são efetuados os testes com maior criticidade até alcançar os menos críticos.

Para a realização do trabalho foi empregado o método da matriz de priorização GUT, que pontua cada caso de teste, com base em uma hipotética ocorrência de adversidade a partir desse caso de teste. São imputadas três notas de 1 a 5, em conformidade com as regras apresentadas, anteriormente na figura 8, para cada um dos parâmetros. Primeiramente é analisada a Gravidade (impactos caso o problema não seja corrigido), posteriormente a urgência (prazo para correção) e por último a tendência (propagação do problema).

Com a conclusão do processo de pontuação dos casos de teste são multiplicadas as três pontuações gerando um valor de grau crítico, permitindo assim a classificação e organização da execução dos casos de teste.

A execução da pontuação dos casos foi analisada e efetuada pelo próprio pesquisador, que possui atuação profissional na área de teste, levantando no final o grau crítico de cada caso de teste. Na figura 15 são expostas as pontuações levantadas para o campo CPF da tela de Identificação do contribuinte.

Figura 15 – Pontuação realizada com base na matriz de priorização GUT

Código Caso de teste	Descrição do caso de teste	Planejado	Executado	Passou	Falhou	Issue (Cód. Erro)	Quem testou	Quando	Tempo executado (min.)	G	U	T	Grau Crítico (GxUxT)
TC 1	Campo CPF - 377.845.609-15									5	5	5	125
TC 2	Campo CPF - 377.845.609-16									4	4	5	80
TC 3	Campo CPF - 999.999.999-99									2	3	3	18
TC 4	Campo CPF - [Sem preenchimento]									4	4	5	80
TC 5	Campo Nome - Romário Maurício									5	4	5	100
TC 6	Campo Nome - Romário Maurício 2									2	2	2	8
TC 7	Campo Nome - Romário Maurício !									2	2	2	8
TC 8	Campo Nome - [Sem preenchimento]									4	4	5	80
TC 9	Campo Tipo de declaração - Ajuste Anual									4	4	5	80
TC 10	Campo Tipo de declaração - Retificadora									4	4	5	80
TC 11	Campo Tipo de declaração - [Sem escolha da opção]									5	4	2	40
TC 12	Campo Nº último recibo - 128471140682									4	4	4	64
TC 13	Campo Nº último recibo - 111111111111									5	5	4	100
TC 14	Campo Nº último recibo - [Sem preenchimento]									2	2	3	12
TC 15	Campo Data Nascimento - 29/02/2000									5	4	5	100
TC 16	Campo Data Nascimento - 31/04/2000									4	5	4	80
TC 17	Campo Data Nascimento - 29/02/2001									4	3	2	24
TC 18	Campo Data Nascimento - [Sem preenchimento]									2	2	3	12
TC 19	Campo Título eleitoral - 4356870906									4	4	4	64
TC 20	Campo Título eleitoral - 4356870907									4	4	3	48
TC 21	Campo Título eleitoral - [Sem preenchimento]									2	2	3	12
TC 22	Campo Tipo de Logradouro - Rua									3	3	4	36
TC 23	Campo Tipo de Logradouro - [Sem escolha da opção]									3	3	2	18
TC 24	Campo Logradouro - Acre									3	3	4	36
TC 25	Campo Logradouro - [Sem preenchimento]									3	3	2	18
TC 26	Campo Número - 2									2	2	3	12
TC 27	Campo Número - [Sem preenchimento]									1	1	1	1
TC 28	Campo Complemento - Apartamento, bloco 2									2	2	3	12
TC 29	Campo Complemento - [Sem preenchimento]									1	1	1	1
TC 30	Campo Bairro - Próspera									2	2	3	12
TC 31	Campo Bairro - [Sem preenchimento]									1	1	1	1
TC 32	Campo UF - SC									3	3	4	36
TC 33	Campo UF - [Sem escolha da opção]									3	3	2	18
TC 34	Campo Município - Criciúma									3	3	4	36
TC 35	Campo Município - [Sem preenchimento]									3	3	2	18
TC 36	Campo CEP - 88813-210									3	3	4	36
TC 37	Campo CEP - [Sem preenchimento]									3	3	2	18
TC 38	Campo DDD - 48									4	3	4	48
TC 39	Campo DDD - [Sem preenchimento]									3	3	2	18
TC 40	Campo Telefone - 3461-2000									4	3	4	48
TC 41	Campo Telefone - [Sem preenchimento]									3	3	2	18
TC 42	Campo Natureza da ocupação - Militar									3	3	4	36
TC 43	Campo Natureza da ocupação - [Sem escolha da opção]									3	3	2	18
TC 44	Campo Ocupação principal - Militar da aeronáutica									3	3	4	36
TC 45	Campo Ocupação principal - [Sem escolha da opção]									3	3	2	18

Fonte: Do autor.

Para a realização da priorização através da matriz de GUT foi empregado o princípio de Pareto conjuntamente a este método, com o objetivo de atingir uma melhor avaliação.

8.1.6 Priorização dos casos utilizando Pareto

O princípio de Pareto tornou-se uma ferramenta muito importante em diversos setores em que se necessita tomar decisões (FERNANDES, 1984), como no caso do trabalho proposto.

No presente trabalho, além de aplicar o método de priorização GUT, será empregado, conjuntamente, o princípio de Pareto com o propósito de alcançar um resultado superior. Para isso serão consideradas as notas obtidas com a aplicação da matriz de priorização GUT e será planejada a execução de 20% dos casos de teste levantados para cada ciclo, sendo que serão priorizados os que possuem maior grau crítico, resultado da multiplicação entre os parâmetros Gravidade, Urgência e Tendência.

Para os ciclos de testes que possuem os mesmos casos de teste levantados, serão desconsiderados os escolhidos nos ciclos anteriores. Além

disso, deve-se considerar que o cálculo da escolha dos 20% dos casos deverá desconsiderar os casos de teste já escolhidos nos ciclos anteriores, diminuindo a quantidade de casos de teste de um ciclo para outro.

Na figura 16 é demonstrada a TRM 1.0 com a seleção de 20% dos casos listados.

Figura 16 – TRM 1.0

Código Caso de teste	Descrição do caso de teste	Planejado	Executado	Passou	Falhou	Issue (Cód. Erro)	Quem testou	Quando	Tempo executado (min.)	G	U	T	Grau Crítico (GxUxT)
TC 1	Campo CPF - 377.845.609-15	X								5	5	5	125
TC 2	Campo CPF - 377.845.609-16	X								4	4	5	80
TC 3	Campo CPF - 999.999.999-99									2	3	3	18
TC 4	Campo CPF - [Sem preenchimento]	X								4	4	5	80
TC 5	Campo Nome - Romário Maurício	X								5	4	5	100
TC 6	Campo Nome - Romário Maurício 2									2	2	2	8
TC 7	Campo Nome - Romário Maurício !									2	2	2	8
TC 8	Campo Nome - [Sem preenchimento]	X								4	4	5	80
TC 9	Campo Tipo de declaração - Ajuste Anual	X								4	4	5	80
TC 10	Campo Tipo de declaração - Retificadora	X								4	4	5	80
TC 11	Campo Tipo de declaração - [Sem escolha da opção]									5	4	2	40
TC 12	Campo Nº último recibo - 128471140682									4	4	4	64
TC 13	Campo Nº último recibo - 111111111111	X								5	5	4	100
TC 14	Campo Nº último recibo - [Sem preenchimento]									2	2	3	12
TC 15	Campo Data Nascimento - 29/02/2000	X								5	4	5	100
TC 16	Campo Data Nascimento - 31/04/2000									4	5	4	80
TC 17	Campo Data Nascimento - 29/02/2001									4	3	2	24
TC 18	Campo Data Nascimento - [Sem preenchimento]									2	2	3	12
TC 19	Campo Título eleitoral - 4356870906									4	4	4	64
TC 20	Campo Título eleitoral - 4356870907									4	4	3	48
TC 21	Campo Título eleitoral - [Sem preenchimento]									2	2	3	12
TC 22	Campo Tipo de Logradouro - Rua									3	3	4	36
TC 23	Campo Tipo de Logradouro - [Sem escolha da opção]									3	3	2	18
TC 24	Campo Logradouro - Acre									3	3	4	36
TC 25	Campo Logradouro - [Sem preenchimento]									3	3	2	18
TC 26	Campo Número - 2									2	2	3	12
TC 27	Campo Número - [Sem preenchimento]									1	1	1	1
TC 28	Campo Complemento - Apartamento, bloco 2									2	2	3	12
TC 29	Campo Complemento - [Sem preenchimento]									1	1	1	1
TC 30	Campo Bairro - Próspera									2	2	3	12
TC 31	Campo Bairro - [Sem preenchimento]									1	1	1	1
TC 32	Campo UF - SC									3	3	4	36
TC 33	Campo UF - [Sem escolha da opção]									3	3	2	18
TC 34	Campo Município - Criciúma									3	3	4	36
TC 35	Campo Município - [Sem preenchimento]									3	3	2	18
TC 36	Campo CEP - 88813-210									3	3	4	36
TC 37	Campo CEP - [Sem preenchimento]									3	3	2	18
TC 38	Campo DDD - 48									4	3	4	48
TC 39	Campo DDD - [Sem preenchimento]									3	3	2	18
TC 40	Campo Telefone - 3461-2000									4	3	4	48
TC 41	Campo Telefone - [Sem preenchimento]									3	3	2	18
TC 42	Campo Natureza da ocupação - Militar									3	3	4	36
TC 43	Campo Natureza da ocupação - [Sem escolha da opção]									3	3	2	18
TC 44	Campo Ocupação principal - Militar da aeronáutica									3	3	4	36
TC 45	Campo Ocupação principal - [Sem escolha da opção]									3	3	2	18

Fonte: Do autor.

Na figura 17 é apresentada a TRM 2.0 com a seleção de 20% dos casos elencados, porém desconsiderando os casos selecionados na TRM 1.0.

Figura 17 – TRM 2.0

Código Caso de teste	Descrição do caso de teste	Planejado	Executado	Passou	Falhou	Issue (Cód. Erro)	Quem testou	Quando	Tempo executado (min.)	G	U	T	Grau Crítico (GxUxT)
TC 3	Campo CPF - 999.999.999-99									2	3	3	18
TC 6	Campo Nome - Romário Maurício 2									2	2	2	8
TC 7	Campo Nome - Romário Maurício !									2	2	2	8
TC 11	Campo Tipo de declaração - [Sem escolha da opção]	X								5	4	2	40
TC 12	Campo Nº último recibo - 128471140682	X								4	4	4	64
TC 14	Campo Nº último recibo - [Sem preenchimento]									2	2	3	12
TC 16	Campo Data Nascimento - 31/04/2000	X								4	5	4	80
TC 17	Campo Data Nascimento - 29/02/2001									4	3	2	24
TC 18	Campo Data Nascimento - [Sem preenchimento]									2	2	3	12
TC 19	Campo Título eleitoral - 4356870906	X								4	4	4	64
TC 20	Campo Título eleitoral - 4356870907	X								4	4	3	48
TC 21	Campo Título eleitoral - [Sem preenchimento]									2	2	3	12
TC 22	Campo Tipo de Logradouro - Rua									3	3	4	36
TC 23	Campo Tipo de Logradouro - [Sem escolha da opção]									3	3	2	18
TC 24	Campo Logradouro - Acre									3	3	4	36
TC 25	Campo Logradouro - [Sem preenchimento]									3	3	2	18
TC 26	Campo Número - 2									2	2	3	12
TC 27	Campo Número - [Sem preenchimento]									1	1	1	1
TC 28	Campo Complemento - Apartamento, bloco 2									2	2	3	12
TC 29	Campo Complemento - [Sem preenchimento]									1	1	1	1
TC 30	Campo Bairro - Próspera									2	2	3	12
TC 31	Campo Bairro - [Sem preenchimento]									1	1	1	1
TC 32	Campo UF - SC									3	3	4	36
TC 33	Campo UF - [Sem escolha da opção]									3	3	2	18
TC 34	Campo Município - Criciúma									3	3	4	36
TC 35	Campo Município - [Sem preenchimento]									3	3	2	18
TC 36	Campo CEP - 88813-210									3	3	4	36
TC 37	Campo CEP - [Sem preenchimento]									3	3	2	18
TC 38	Campo DDD - 48	X								4	3	4	48
TC 39	Campo DDD - [Sem preenchimento]									3	3	2	18
TC 40	Campo Telefone - 3461-2000	X								4	3	4	48
TC 41	Campo Telefone - [Sem preenchimento]									3	3	2	18
TC 42	Campo Natureza da ocupação - Militar									3	3	4	36
TC 43	Campo Natureza da ocupação - [Sem escolha da opção]									3	3	2	18
TC 44	Campo Ocupação principal - Militar da aeronáutica									3	3	4	36
TC 45	Campo Ocupação principal - [Sem escolha da opção]									3	3	2	18

Fonte: Do autor.

Com a priorização e escolha dos casos de testes concluída, sucede-se a execução dos casos para verificar o comportamento obtido e descrever as informações necessárias a cada item.

8.1.7 Execução dos casos de teste

Após a seleção dos casos de teste foi verificado que os casos com menor prioridade ficaram de fora dos ciclos de teste, dando a entender que não há a necessidade de executá-los.

Todavia, para os casos de teste planejados, a execução foi realizada. Para os testes que apresentaram resultado positivo, a coluna *Passou* da planilha foi preenchida com um *x*. Em contrapartida, foi preenchida a coluna *Falhou* para os que não resultaram em um comportamento esperado, ou seja, apresentaram adversidades em relação ao que foi anotado na descrição do caso de teste. Além disso, deve-se registrar a adversidade ocorrida em uma ferramenta que reporte as falhas, preenchendo o *Issue* (Código do erro) em conjunto com as colunas *Data*, *Quem testou* e o tempo consumido para a execução do caso de teste.

A figura 18 mostra a TRM 1.0 com todos os casos planejados executados e com as informações fundamentais preenchidas de acordo com o resultado exposto.

Figura 18 – TRM 1.0 após a execução dos casos e preenchimento das devidas informações

Código Caso de teste	Descrição do caso de teste	Planejado	Executado	Passou	Falhou	Issue (Cód. Erro)	Quem testou	Quando	Tempo executado (min.)	G	U	T	Grau Crítico (GxUxT)
TC 1	Campo CPF - 377.845.609-15	X	X	X						5	5	5	125
TC 2	Campo CPF - 377.845.609-16	X	X	X						4	4	5	80
TC 3	Campo CPF - 999.999.999-99									2	3	3	18
TC 4	Campo CPF - [Sem preenchimento]	X	X	X						4	4	5	80
TC 5	Campo Nome - Romário Maurício	X	X	X						5	4	5	100
TC 6	Campo Nome - Romário Maurício 2									2	2	2	8
TC 7	Campo Nome - Romário Maurício !									2	2	2	8
TC 8	Campo Nome - [Sem preenchimento]	X	X	X						4	4	5	80
TC 9	Campo Tipo de declaração - Ajuste Anual	X	X	X						4	4	5	80
TC 10	Campo Tipo de declaração - Retificadora	X	X	X						4	4	5	80
TC 11	Campo Tipo de declaração - [Sem escolha da opção]									5	4	2	40
TC 12	Campo Nº último recibo - 128471140682									4	4	4	64
TC 13	Campo Nº último recibo - 111111111111	X	X	X						5	5	4	100
TC 14	Campo Nº último recibo - [Sem preenchimento]									2	2	3	12
TC 15	Campo Data Nascimento - 29/02/2000	X	X	X						5	4	5	100
TC 16	Campo Data Nascimento - 31/04/2000									4	5	4	80
TC 17	Campo Data Nascimento - 29/02/2001									4	3	2	24
TC 18	Campo Data Nascimento - [Sem preenchimento]									2	2	3	12
TC 19	Campo Título eleitoral - 4356870906									4	4	4	64
TC 20	Campo Título eleitoral - 4356870907									4	4	3	48
TC 21	Campo Título eleitoral - [Sem preenchimento]									2	2	3	12
TC 22	Campo Tipo de Logradouro - Rua									3	3	4	36
TC 23	Campo Tipo de Logradouro - [Sem escolha da opção]									3	3	2	18
TC 24	Campo Logradouro - Acre									3	3	4	36
TC 25	Campo Logradouro - [Sem preenchimento]									3	3	2	18
TC 26	Campo Número - 2									2	2	3	12
TC 27	Campo Número - [Sem preenchimento]									1	1	1	1
TC 28	Campo Complemento - Apartamento, bloco 2									2	2	3	12
TC 29	Campo Complemento - [Sem preenchimento]									1	1	1	1
TC 30	Campo Bairro - Próspera									2	2	3	12
TC 31	Campo Bairro - [Sem preenchimento]									1	1	1	1
TC 32	Campo UF - SC									3	3	4	36
TC 33	Campo UF - [Sem escolha da opção]									3	3	2	18
TC 34	Campo Município - Criciúma									3	3	4	36
TC 35	Campo Município - [Sem preenchimento]									3	3	2	18
TC 36	Campo CEP - 88813-210									3	3	4	36
TC 37	Campo CEP - [Sem preenchimento]									3	3	2	18
TC 38	Campo DDD - 48									4	3	4	48
TC 39	Campo DDD - [Sem preenchimento]									3	3	2	18
TC 40	Campo Telefone - 3461-2000									4	3	4	48
TC 41	Campo Telefone - [Sem preenchimento]									3	3	2	18
TC 42	Campo Natureza da ocupação - Militar									3	3	4	36
TC 43	Campo Natureza da ocupação - [Sem escolha da opção]									3	3	2	18
TC 44	Campo Ocupação principal - Militar da aeronáutica									3	3	4	36
TC 45	Campo Ocupação principal - [Sem escolha da opção]									3	3	2	18

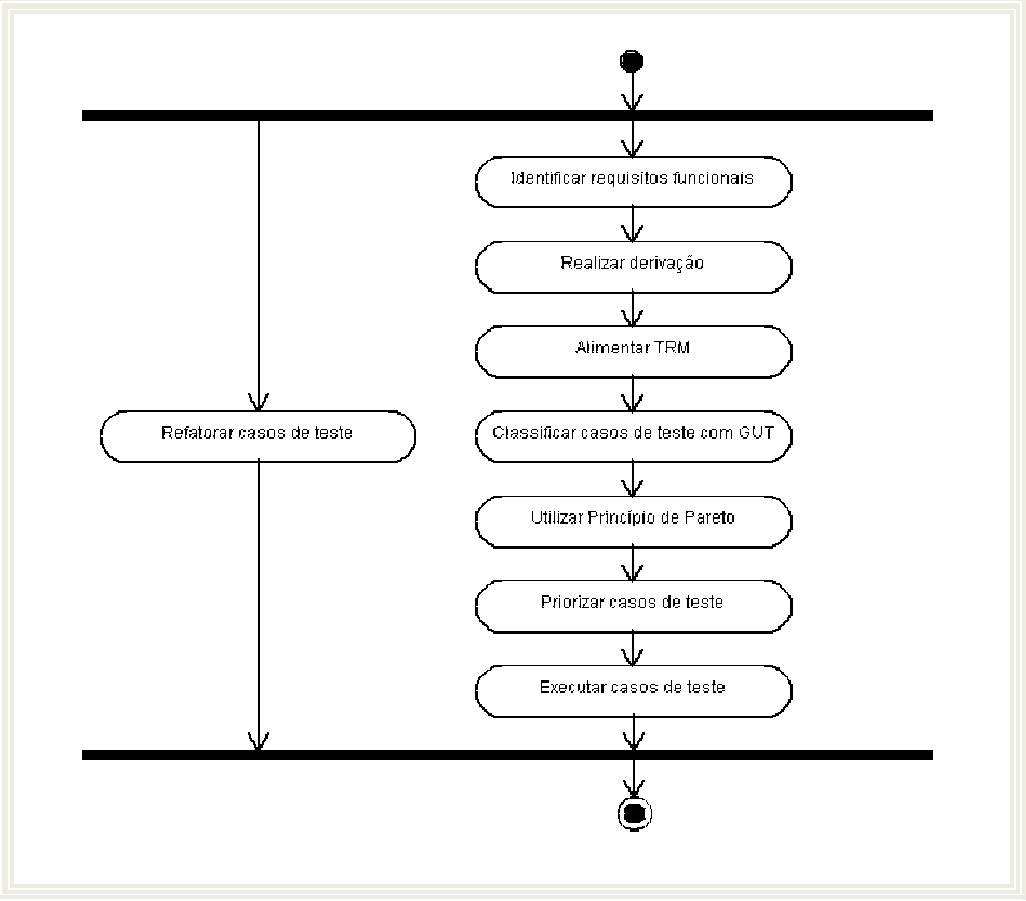
Fonte: Do autor.

A etapa de explanação da metodologia chega ao final com a execução dos casos de teste. Concluindo, assim, a criação do método de priorização dos casos de teste.

8.2 RESULTADOS OBTIDOS

O estudo atingiu os objetivos desejados em relação à criação de um método de priorização dos casos de teste. A figura 19 apresenta um diagrama de atividades com as etapas executadas a fim de esclarecer o resultado obtido para cada item, sendo:

Figura 19 – Diagrama de atividades do trabalho proposto



Fonte: Do autor.

a) **identificar requisitos funcionais:** foram expostas em detalhes as funções de cada campo como as entradas, saídas e exceções, das telas de Identificação do Contribuinte e do Cadastro de Dependentes;

b) **realizar derivação:** diante dos requisitos levantados, foram levantados os casos de teste, separados para cada campo, que historicamente mostram-se mais propensos a detectar falhas;

c) **alimentar TRM:** foi realizada a alimentação da TRM com os casos de teste levantados utilizando a derivação funcional;

d) **classificar casos de teste com GUT:** obteve os valores em relação a Gravidade, Urgência e Tendência de cada caso de teste, permitindo que fossem classificados os testes mais críticos em relação ao menos cruciais;

e) **utilizar Princípio de Pareto:** resultou na priorização de 20% dos casos de testes derivados, utilizando-se da pontuação realizada através da métrica de GUT;

f) **priorizar casos de teste:** obteve uma listagem de casos de teste planejados após a utilização da métrica de GUT e do Princípio 80/20;

g) **executar casos de teste:** procedeu na análise dos casos de testes planejados que foram executados;

h) **refatorar casos de teste:** caso alguma falha seja encontrada na realização do fluxo principal, deve-se executar novamente os casos de teste que encontraram erros.

A figura 20 contém o processo validado, expondo os percentuais dos casos de testes planejados, executados, os que possuíram resultado positivo e os que apresentaram falhas.

Figura 20 – Processo validado – Software IRPF

Processo validado						
	TRM 1.0	TRM 2.0	TRM 3.0	TRM 4.0	TRM 5.0	TRM 6.0
Quantidade planejada	20%	20%	20%	20%	20%	20%
Quantidade executada	20%	20%	20%	20%	20%	20%
Quantidade "Passou"	100%	100%	100%	100%	100%	100%
Quantidade "Falhou"	0%	0%	0%	0%	0%	0%

Fonte: Do autor.

Os percentuais em relação à *Quantidade planejada* e a *Quantidade executada* possuíram os valores conforme a métrica apresentada na matriz de priorização GUT associada ao Princípio de Pareto e a técnica de derivação de conhecimento e experiência do especialista de teste.

No resultado da execução, *Quantidade "Passou"* e *Quantidade "Falhou"*, pode-se perceber que não foram encontradas falhas, neste caso, considera-se que o software de Declaração do Imposto de Renda de Pessoa Física da Receita Federal é um sistema estável, que já passou por diversas baterias de testes antes de ser liberado ao usuário, condição imprescindível para um programa desse porte. Seria, portanto, inviável disponibilizá-lo apresentando erros, visto que uma falha poderia acarretar em atrasos na

entrega do imposto pelo contribuinte ou, ainda, permitir seu envio com o preenchimento incorreto.

Foi constatado que o software, da Receita Federal, escolhido no início do desenvolvimento do trabalho proposto possui uma grande estabilidade, já que são implementadas poucas melhorias de uma versão para outra, permitindo assim que seja dada atenção aos problemas existentes e consequentemente realizando sua correção.

Portanto, para proceder com a validação da criação da técnica para priorização dos casos de teste foi necessário escolher outro software que não fosse estável como o software da Receita Federal. Foi realizada pesquisa no site da Anvisa, procurando um serviço que fosse disponibilizado a qualquer usuário e escolheu-se o Formulário para denúncias, reclamações e solicitações[3].

Neste software, também foram realizados todos os passos, desde o levantamento dos casos de teste utilizando a técnica de derivação, passando pela classificação utilizando GUT e pela priorização utilizando o Princípio de Pareto até a execução dos casos de teste.

A figura 21 compreende o processo validado, apresentando os percentuais dos casos de testes planejados, executados, os que possuíram resultado positivo e os que apresentaram falhas.

Figura 21 – Processo validado – Software Formulário para denúncias, reclamações e solicitações

Processo validado						
	TRM 1.0	TRM 2.0	TRM 3.0	TRM 4.0	TRM 5.0	TRM 6.0
Quantidade planejada	20%	20%	20%	20%	20%	20%
Quantidade executada	20%	20%	20%	20%	20%	20%
Quantidade "Passou"	25%	100%	40%	25%	100%	100%
Quantidade "Falhou"	75%	0%	60%	75%	0%	0%

Fonte: Do autor.

Com o processo validado após a aplicação dos testes no segundo software, pode-se constatar que os objetivos propostos no inicio da pesquisa

[3] Disponível em <http://www10.anvisa.gov.br/ouvidoria/CadastroProcedimentoInternetACT.do?metodo=inicia>. Acesso em 31 mai. 2015.

foram alcançados, proporcionando como resultado final uma técnica de priorização dos casos de teste que continua garantindo a qualidade do sistema, porém permitindo a redução do tempo consumido para a realização dessa atividade.

9 CONCLUSÃO

As empresas sempre se empenham em fornecer um produto de qualidade para seus consumidores, no caso de uma empresa de software isso não é diferente. O item principal para estabelecer a qualidade de um sistema é o teste de software e é por meio dele que serão analisados os comportamentos do sistema, para avaliar se realmente foram alcançados os resultados esperados.

Comprovou-se que a utilização de métodos de priorização dos casos de teste proporciona um aproveitamento superior do tempo destinado para esta etapa, uma vez que se almeja optar pela execução dos casos de teste em pontos do sistema, que historicamente, apresentam adversidades.

Com esse objetivo que foi elaborado o processo de priorização de casos de testes, a fim de validar as funcionalidades dos softwares. Assim, antes de iniciar a execução dos testes, é possível classificar os casos de testes que possuem maior propensão a detectar erros, podendo priorizá-los.

Para alcançar o estabelecido nesse trabalho, foi adquirido conhecimento na área de teste de software, principalmente, em relação aos conceitos e técnicas. Utilizando o critério de derivação baseado no conhecimento e experiência do especialista foram levantados os casos de teste. Posteriormente, o foco foi concentrado na classificação dos casos de teste utilizando a matriz de priorização GUT e planejando os casos a serem executados a partir da priorização, utilizando o princípio 80/20 de Pareto.

Como resultado final, foi criado um processo baseado na matriz de rastreabilidade aplicado à técnica de priorização de casos de teste, que foram registradas em planilha eletrônica, na qual os casos de teste são detalhados, possibilitando sua utilização em futuras consultas. Desta maneira, as informações documentadas servirão como um histórico para os testadores. Posto isto, permite-se concluir que o trabalho conseguiu alcançar seu objetivo geral.

Os estudos realizados neste trabalho proporcionaram um ganho profissional ao pesquisador, visto que obteve compreensão dos conceitos das técnicas que utiliza no dia-a-dia, podendo praticá-las assimilando seu

funcionamento também na teoria. Além disso, teve a oportunidade de agregar esses métodos na sua atividade profissional, apresentando melhoria na qualidade dos seus testes e otimizando o seu tempo de forma mais satisfatória.

Dando continuidade a esta pesquisa, tem-se como possibilidade de trabalhos futuros a:

a) criação de outra técnica de priorização dos casos de teste para fazer comparação ao criado;
b) implementação dessa metodologia em Java;
c) ampliação das variáveis de ambiente utilizadas na TRM;
d) aumento da cobertura dos requisitos funcionais e, principalmente, dos requisitos não funcionais;
e) implantação dessa metodologia em uma empresa de software, garantindo uma validação ainda mais confiável.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 10520**: apresentação de citações em documentos: procedimento. Rio de Janeiro, 1988.

AVIZIENIS, A. et al**.** Basic Concepts and Taxonomy of Dependable and Secure Computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n.1, p. 11–33, 2004. Disponível em:<http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1335465>. Acesso em 22 nov. 2014.

BASTOS, Anderson et al. **Base de conhecimento em teste de software.** 2.ed. Niterói. Martins Fontes, 2007.

BEHR, Ariel; MORO, Eliane Lourdes da Silva; ESTABEL, Lizandra Brasil. **Gestão da biblioteca escolar:** metodologias, enfoques e aplicação de ferramentas de gestão e serviços de biblioteca. SciELO Scientific Electronic Library Online, v. 37, n.2, p. 32-42, 2008. Disponível em: <http://dx.doi.org/10.1590/S0100-19652008000200003>

CARTAXO, Emanuela Gadelha. **Geração de casos de teste funcional para aplicações de celulares**. 2006. 146 f. Dissertação (Mestrado) - Ufcg, Campina Grande, 2006. Disponível em: <http://docs.computacao.ufcg.edu.br/posgraduacao/dissertacoes/2006/Dissertacao_EmanuelaGCartaxo.pdf>. Acesso em: 20 out. 2014.

CAVALCANTI, Tilene Corsi. **A importância do processo de teste para a qualidade do software.** 81 f. Monografia (Curso de Tecnologia em Informática para Gestão de Negócios)-Faculdade de Tecnologia da Zona Leste-FATECZL. São Paulo, 2009. Disponível em:<http://fateczl.edu.br/TCC/2009-2/tcc-64.pdf>. Acesso em: 22 nov. 2014.

CRAIG, R.D., JASKIEL, S. P. **Systematic Software Testing**. Artech House Publishers, Boston, 2002.

CRESPO, Adalberto Nobiato et al. **Uma metodologia para teste de software no contexto da melhoria de processo.** 15 f. Artigo-Centro de Pesquisas Renato Archer (CENPRA). Campinas, 2012. Disponível em:<http://www.lbd.dcc.ufmg.br/colecoes/sbqs/2004/024.pdf>. Acesso em: 22 nov. 2014.

DAMIANI, Camilla. **Desenvolvimento de uma metodologia baseada na automação do processo de testes de software como estratégia para a garantia da cobertura funcional.** 84 f. Monografia (Graduação em Ciência da Computação)-Universidade do Extremo Sul Catarinense-UNESC. Criciúma, 2012. Disponível em:<http://tcc.kironunesc.net.br/arquivos/trabalhos/337.pdf>. Acesso em: 22 nov. 2014.

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

DIAS, Nauane Karoline Brazolino; ARAÚJO Marco Antônio Pereira. Boas Práticas na Engenharia de Requisitos. **Engenharia de software magazine.** Rio de Janeiro, ano 3, n. 27, p.14-21, 2010.

DIAS NETO, Arilo Claudio. **Introdução a Teste de Software**. Qualidade de Software: Entenda os principais conceitos sobre Testes e Inspeção de Software. São Paulo, 2007.

FERNANDES, José Carlos de F. Administração de material um enfoque sistêmico. 2 ed. Rio de Janeiro: Livros Técnicos e Científicos, 1984.

FEWSTER, Mark, GRAHAM, Dorothy. **Software Test Automation.** Addison-Wesley Professional, 1999.

GOTEL, O. e FINKELSTEIN, A. (1995) "**Contribution Structures**", Proceedings of 2nd Internationak Symposium on Requeriments Engineering. IEEE Computer Society Press.

GRINWALD, Raanan et al. **User defined coverage** - a tool supported methodology for design verification. In Proceedings of the 35th Design Automation Conference, pages 158–165, June 1998.

GUSTAFSON, David A. **Engenharia de Software.** Porto Alegre: Bookman, 2002.

IEEE. **IEEE Recommended Practice for Software Requirements Specifications**. New York, out. 1998.

INTHURN, Cândida. **Qualidade & teste de software:** engenharia de software, qualidade de software, qualidade de produtos de software, teste de software, formalização do processo de teste, aplicação prática dos testes. Florianópolis: Visual Books, 2001.

JORGENSEN, Paul C. **Software Testing** – a Craftsman Approach. CRC Press,1995. 272 p.

KOCH, Richard. **O princípio 80/20:** O segredo de se realizar mais com menos. Rocco, 2006

LAPLANTE, Phillip A., Requirements Engineering for Software and Systems, Second Edition, Taylor & Francis, 2014

MARSHALL JUNIOR, Isnard; CIERCO, Agliberto Alves; ROCHA, MOTA, Edmarson Bacelar; LEUSIN, Sérgio. **Gestão da qualidade.** 9 ed. Rio de Janeiro: FGV, 2008.

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

MYERS, Glenford J. **The Art of Software Testing**. New York: John Wiley & Sons, Second Edition, 2004.

MYERS, Glenford J., John Wiley & Sons, **The Art of Software Testing**, 2 ed. Nova Jérsei: 2004.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software:** fundamentos, métodos e padrões. São Paulo: LTC Editora, 2000.

PRESSMAN, R. S. **Engenharia de Software**. 5 ed. Rio de Janeiro: McGraw-Hill, 843 p., 2002.

_____. **Software Engineering:** A Practitioner's Approach. 5 ed. New York: McGraw-Hill, 2001.

_____. **Software Engineering:** A Practitioner's Approach. Makron Books, 2009.

_____. **Engenharia de Software.** 6 ed. São Paulo: McGraw-Hill, 2006.

REINERT, Cleverson. **Utilização de testes automatizados para ampliar a cobertura funcional na manutenção de software.** 118 f. Monografia (Graduação em Ciência da Computação)-Universidade do Extremo Sul Catarinense-UNESC. Criciúma, 2012. Disponível em:<http://tcc.kironunesc.net.br/arquivos/trabalhos/336.pdf>. Acesso em: 22 nov. 2014.

RIOS, Emerson. **Documentação de Teste.** Dissecando a norma ou padrão IEEE 829. Instituto de Teste de Software, 2007. Disponível em: <www.proimpe.org.br/portal/spic/bco_arq/Documentacao_de_teste.doc>. Acesso em: 21 nov. 2014.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software:** Segunda edição revisada e ampliada. 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.

SHERBA, S.; ANDERSON, K. **A framework for managing traceability relationships between requirements and architectures**. Proceedings of the Second International Workshop From Software Requirements to Architecture, 2003.

SILVA, Karina Rocha Gomes da. **Uma Metodologia de Verificação Funcional para Circuitos Digitais.** 2007. 121 f. Tese (Doutorado) - Ufcg, Campina Grande, 2007. Disponível em: <http://lad.dsc.ufcg.edu.br/uploads/Lad/tese_karina.pdf>. Acesso em: 10 out. 2014.

SIQUEIRA. Anderson José. **Ferramenta:** ConDado. ATIFS. Disponível em: <http://www.inpe.br/atifs/ferramentas/ferramenta_condado.php>. Acesso em 09 nov. 2014.

SOMMERVILLE, Ian. **Engenharia de software.** São Paulo: Addison-Wesley, 2003.

SOMMERVILLE, I., **Engenharia de Software**, 8 ed. São Paulo: Pearson – Addison Wesley, 2007.

SOMMERVILLE, Ian. **Software Engineering**. 9 ed. Massachusetts: Pearson, 2001.

SPANOUDAKIS, G.; ZISMAN, A. **Software traceability:** A roadmap. In S. Chang (Eds.), Advances in Software Engineering and Knowledge: Recent Advances, Vol III. World Scientific Publishing, 2005

YOURDON, Edward. **Análise estruturada moderna.** 10. ed. Rio de Janeiro: Campus, 1990. 836p.

APÊNDICE(S)

APÊNDICE A – CASOS DE TESTE LEVANTADOS PARA O SITE DA IRPF 2015

Casos de teste (Criar nova declaração)	
CPF	**Saída esperada**
377.845.609-15	Aceitar CPF
377.845.609-16	Erro: Número de inscrição no CPF inválido
999.999.999-99	Erro: Número de inscrição no CPF inválido
[sem preenchimento]	Erro: O campo indicativo "Número de inscrição no CPF" não foi informado

Casos de teste (Criar nova declaração)	
Nome	**Saída esperada**
Romário Maurício	Aceitar nome
Romário Maurício 2	Erro: Nome não pode ter números
Romário Maurício !	Erro: Nome tem caracteres inválidos
[sem preenchimento]	Erro: O campo indicativo "Nome" não foi informado

Casos de teste (Ident. do contribuinte)	
Tipo de declaração	**Saída esperada**
Ajuste Anual	Aceitar opção escolhida
Retificadora	Aceitar opção escolhida
[sem escolha da opção]	Erro: O campo indicativo "Que tipo de declaração você deseja fazer?" não foi informado

Casos de teste (Ident. do contribuinte)	
Nº último recibo	**Saída esperada**
128471140682	Aceitar número do último recibo
111111111111	Erro: Nº do recibo inválido
[sem preenchimento]	Advertência: caso tenha sido entregue a declaração do ano anterior, deve-se informar o número do recibo

Casos de teste (Ident. do contribuinte)	
Data Nasc	**Saída esperada**
29/02/2000	Aceitar data de nascimento
31/04/2000	Erro: Data de nascimento inválida
29/02/2001	Advertência: Verificar dia, mês e ano de nascimento, pois o ano não é bissexto
[sem preenchimento]	Erro: Data de nascimento não informada

Casos de teste (Ident. do contribuinte)	
Título eleitoral	**Saída esperada**
4356870906	Aceitar Título eleitoral
4356870907	Erro: O campo indicativo "Título eleitoral" não foi informado
[sem preenchimento]	Advertência: O campo indicativo "Título eleitoral" não foi informado

Casos de teste (Ident. do contribuinte)	
Tipo de Logradouro	**Saída esperada**
Rua	Aceitar opção escolhida
[sem escolha da opção]	Erro: O campo indicativo "Tipo de logradouro" não foi informado

Casos de teste (Ident. do contribuinte)	
Logradouro	**Saída esperada**
Acre	Aceitar opção escolhida
[sem preenchimento]	Erro: O campo indicativo "Logradouro" não foi informado

Casos de teste (Ident. do contribuinte)	
Número	**Saída esperada**
2	Aceitar número
[sem preenchimento]	Aceitar número em branco

Casos de teste (Ident. do contribuinte)	
Complemento	**Saída esperada**
Apartamento, bloco 2	Aceitar complemento
[sem preenchimento]	Aceitar complemento em branco

Casos de teste (Ident. do contribuinte)	
Bairro	**Saída esperada**
Próspera	Aceitar Bairro
[sem preenchimento]	Aceitar Bairro em branco

Casos de teste (Ident. do contribuinte)	
UF	**Saída esperada**
SC	Aceitar opção escolhida
[sem escolha da opção]	Erro: O campo indicativo "UF" não foi informado

Casos de teste (Ident. do contribuinte)	
Município	**Saída esperada**
Criciúma	Aceitar opção escolhida
[sem escolha da opção]	Erro: O campo indicativo "Município" não foi informado

Casos de teste (Ident. do contribuinte)	
CEP	**Saída esperada**
88813-210	Aceitar opção escolhida
[sem preenchimento]	Erro: O campo indicativo "CEP" não foi informado. Informe-o com 8 caracteres numéricos

Casos de teste (Ident. do contribuinte)	
DDD	**Saída esperada**
48	Aceitar DDD
[sem preenchimento]	Aceitar DDD em branco

Casos de teste (Ident. do contribuinte)	
Telefone	**Saída esperada**
3461-2000	Aceitar telefone
[sem preenchimento]	Aceitar telefone em branco

Casos de teste (Ident. do contribuinte)	
Natureza da ocupação	**Saída esperada**
Militar	Aceitar opção escolhida
[sem escolha da opção]	Erro: O campo indicativo "Natureza da Ocupação" não foi informado.

Casos de teste (Ident. do contribuinte)	
Ocupação principal	**Saída esperada**
Militar da aeronáutica	Aceitar opção escolhida
[sem escolha da opção]	Erro: O campo indicativo "Natureza da Ocupação" não foi informado.

Casos de teste (Dependentes)	
Tipo de dependente	**Saída esperada**
Filho(a) ou enteado(a) até 21 anos	Aceitar opção escolhida
[sem escolha da opção]	Erro: O campo indicativo "Tipo de dependente" não foi informado.

Casos de teste (Dependentes)	
Nome	**Saída esperada**
Dependente	Aceitar nome
Dependente 1	Erro: Nome não pode ter números
Dependente $	Erro: Nome tem caracteres inválidos
[sem preenchimento]	Erro: O campo indicativo "Nome" não foi informado

Casos de teste (Dependentes)	
CPF	**Saída esperada**
377.845.609-15	Aceitar CPF
377.845.609-16	Erro: O campo indicativo "CPF do dependente" está inválido
111.111.111-11	Erro: O campo indicativo "CPF do dependente" está inválido
[sem preenchimento]	Aceitar CPF

Casos de teste (Dependentes)	
Data Nasc	**Saída esperada**
29/02/2000	Aceitar data de nascimento
31/04/2000	Erro: O campo indicativo "Data de Nascimento" está inválido
29/02/2001	Advertência: Verificar dia, mês e ano de nascimento, pois o ano não é bissexto
[sem preenchimento]	Erro: Data de nascimento não informada

APÊNDICE B – CASOS DE TESTE LEVANTADOS PARA O SITE DA ANVISA

Casos de teste (Dados para contato)	
Tipo do contato	**Saída esperada**
Pessoa Física	Aceitar CPF
Clicar no botão ?	Deve abrir uma página indicando as informações para preenchimento

Casos de teste (Dados para contato)	
Nome	**Saída esperada**
Romário Maurício	Aceitar nome
Romário Maurício 2	Erro: Nome não pode ter números
Romário Maurício !	Erro: Nome tem caracteres inválidos
[sem preenchimento]	Erro: O campo indicativo "Nome" não foi informado

Casos de teste (Dados para contato)	
Cor ou raça	**Saída esperada**
Parda	Aceitar opção escolhida
Selecione	Deve ser preenchido, pois tem uma opção "Não informado".

Casos de teste (Dados para contato)	
CPF/CNPJ	**Saída esperada**
377.845.609-15	Aceitar CPF
377.845.609-16	Não permitir gravar, pois é um CPF com DV incorreto
111.111.111-11	Não permitir gravar, pois é um CPF inválido
[sem preenchimento]	Aceitar CPF

Casos de teste (Dados para contato)	
Endereço	**Saída esperada**
Acre	Aceitar opção escolhida
[sem preenchimento]	Aceitar Endereço em branco

Casos de teste (Dados para contato)	
Bairro	**Saída esperada**
Próspera	Aceitar Bairro
[sem preenchimento]	Aceitar Bairro em branco

Casos de teste (Dados para contato)	
Estado	**Saída esperada**
SC	Aceitar opção escolhida
Conferência das opções existentes	O campo "Estado" não foi informado
[sem escolha da opção]	Deve possuir os 27 estados do Brasil

Casos de teste (Dados para contato)	
Cidade	**Saída esperada**
Criciúma	Aceitar opção escolhida
[sem escolha da opção]	O campo indicativo "Município" não foi informado

Casos de teste (Dados para contato)	
CEP	**Saída esperada**
88813-210	Aceitar opção escolhida
[sem preenchimento]	Erro: O campo indicativo "CEP" não foi informado. Informe-o com 8 caracteres numéricos

Casos de teste (Dados para contato)	
Fone	**Saída esperada**
(48)3461-2000	Aceitar telefone
[sem preenchimento]	O campo indicativo "Fone" não foi informado

Casos de teste (Dados para contato)	
Fax	**Saída esperada**
(48)3461-2000	Aceitar telefone
[sem preenchimento]	O campo indicativo "Fone" não foi informado

Casos de teste (Dados para contato)	
Fone 2	**Saída esperada**
(48)3461-2000	Aceitar telefone
[sem preenchimento]	O campo indicativo "Fone" não foi informado

Casos de teste (Dados para contato)	
Fone 2	**Saída esperada**
(48)3461-2000	Aceitar telefone
[sem preenchimento]	O campo indicativo "Fone" não foi informado

Casos de teste (Dados para contato)	
E-mail	**Saída esperada**
teste@tcc.com.br	Aceitar número
[sem preenchimento]	O campo indicativo "E-mail" não foi informado

Casos de teste (Dados do Reclamado/Denunciado)	
Tipo do reclamado	**Saída esperada**
Pessoa Física	Aceitar CPF
Clicar no botão ?	Deve abrir uma página indicando as informações para preenchimento

Casos de teste (Dados do Reclamado/Denunciado)	
Nome	**Saída esperada**
Romário Maurício	Aceitar nome
Romário Maurício 2	Erro: Nome não pode ter números
Romário Maurício !	Erro: Nome tem caracteres inválidos
[sem preenchimento]	Erro: O campo indicativo "Nome" não foi informado

Casos de teste (Dados do Reclamado/Denunciado)	
CPF/CNPJ	**Saída esperada**
377.845.609-15	Aceitar CPF
377.845.609-16	Não permitir gravar, pois é um CPF com DV incorreto
111.111.111-11	Não permitir gravar, pois é um CPF inválido
[sem preenchimento]	Aceitar CPF

Casos de teste (Dados do Reclamado/Denunciado)	
Endereço	**Saída esperada**
Acre	Aceitar opção escolhida
[sem preenchimento]	Aceitar Endereço em branco

Casos de teste (Dados do Reclamado/Denunciado)	
Bairro	**Saída esperada**
Próspera	Aceitar Bairro
[sem preenchimento]	Aceitar Bairro em branco

Casos de teste (Dados do Reclamado/Denunciado)	
Estado	**Saída esperada**
SC	Aceitar opção escolhida
[sem escolha da opção]	O campo "Estado" não foi informado
Opções existentes	Deve possuir os 27 estados do Brasil

Casos de teste (Dados do Reclamado/Denunciado)	
Cidade	**Saída esperada**
Criciúma	Aceitar opção escolhida
[sem escolha da opção]	O campo indicativo "Município" não foi informado

Casos de teste (Dados do Reclamado/Denunciado)	
CEP	**Saída esperada**
88813-210	Aceitar opção escolhida
[sem preenchimento]	Erro: O campo indicativo "CEP" não foi informado. Informe-o com 8 caracteres numéricos

Casos de teste (Dados do Reclamado/Denunciado)	
Fone	**Saída esperada**
(48)3461-2000	Aceitar telefone
[sem preenchimento]	O campo indicativo "Fone" não foi informado

Casos de teste (Dados do Reclamado/Denunciado)	
E-mail	**Saída esperada**
teste@tcc.com.br	Aceitar número
[sem preenchimento]	O campo indicativo "E-mail" não foi informado

Casos de teste (Dados do procedimento)	
Assunto	**Saída esperada**
Teste	Aceitar opção escolhida
[sem preenchimento]	Aceitar Endereço em branco

Casos de teste (Dados do procedimento)	
Nº Processo ou Expediente Anterior 1	**Saída esperada**
1234	Aceitar número
abcd	Não aceitar letras
[sem preenchimento]	Aceitar sem preenchimento

Casos de teste (Dados do procedimento)	
Nº Processo ou Expediente Anterior 2	**Saída esperada**
1234	Aceitar número
abcd	Não aceitar letras
[sem preenchimento]	Aceitar sem preenchimento

Casos de teste (Dados do procedimento)	
Nº Processo ou Expediente Anterior 3	**Saída esperada**
1234	Aceitar número
abcd	Não aceitar letras
[sem preenchimento]	Aceitar sem preenchimento

Casos de teste (Dados do procedimento)	
Nº Demandas Ouvidori@tende Anterior 1	**Saída esperada**
1234	Aceitar número
abcd	Não aceitar letras
[sem preenchimento]	Aceitar sem preenchimento

Casos de teste (Dados do procedimento)	
Nº Demandas Ouvidori@tende Anterior 2	**Saída esperada**
1234	Aceitar número
abcd	Não aceitar letras
[sem preenchimento]	Aceitar sem preenchimento

Casos de teste (Dados do procedimento)	
Nº Demandas Ouvidori@tende Anterior 3	**Saída esperada**
1234	Aceitar número
abcd	Não aceitar letras
[sem preenchimento]	Aceitar sem preenchimento

Casos de teste (Dados do procedimento)	
Resumo	**Saída esperada**
Teste	Aceitar opção escolhida
[sem preenchimento]	Aceitar Endereço em branco

APÊNDICE C – ARTIGO

Construção De Um Processo De Priorização De Casos De Testes

Romário Antônio Maurício Júnior[1]

[1]Ciência da Computação – Universidade do Extremo Sul Catarinense (UNESC)
Criciúma – SC – Brasil

romario_juninho_@hotmail.com

Abstract. *Atualmente é essencial a realização de testes nos sistemas, para garantir a sua qualidade. Adequado a esse cenário, este estudo apresenta a criação de uma técnica para priorização dos casos de testes feitos no sistema para ampliar a validação das funcionalidades do software*

Resumo. *Currently is essential the realization for tests on systems, to ensure their quality. Suitable for this scenario, this study presents the creating of a technique for priorization of tests cases made on the system to extend the validation of the functionality of the software.*

1 Introdução

A utilização dos testes nos sistemas é de suma importância para garantir a qualidade do mesmo e a satisfação do cliente em utilizar um software que possua poucos erros e que não influencie negativamente em seu trabalho.

Segundo Dias Neto (2007), teste de software é o método de execução de um sistema para definir se ele alcançou suas especificações e funcionou de modo correto no ambiente para o qual foi planejado. Nesse processo, têm-se como objetivo encontrar a maior quantidade de falhas possíveis, visando maior confiança no produto fornecido ao cliente.

Conforme Pressman (2006), o software está em constante evolução e apresentará defeitos durante toda a sua existência, devido às frequentes alterações realizadas no sistema para se adequar ao que o cliente necessita, ou seja, quanto mais alterações são realizadas no sistema, maior será a probabilidade de surgirem novas falhas, que acarretam na queda da qualidade do software.

O tempo destinado a execução de testes é curto. Para que, isso não resulte em uma queda na qualidade de um sistema são utilizados critérios para derivar os casos de testes que serão executados (DIAS NETO, 2014). Existe o critério de particionamento em classes de equivalência, que de acordo com

Rocha et al (2001) tem por objetivo diminuir a quantidade de casos de teste, fazendo a seleção de apenas um caso de teste de cada classe, considerando que todos os elementos de uma classe devem possuir o mesmo comportamento. Outro critério que pode ser utilizado é o critério de análise do valor limite, que conforme Pressman (2005) um grande número de erros tende a ocorrer nos limites dos valores do domínio de entrada de cada classe. Existe também o critério Grafo de causa-efeito, que segundo Rocha et al (2001), verifica o efeito combinado de dados de entrada, identificando as causas e os efeitos para então tomar a decisão de quais casos de testes serão executados.

Além desses, de acordo com Dias Neto (2014), outros critérios podem ser utilizados dependendo da necessidade. Dentre estes critérios estão: teste de desempenho, teste de usabilidade, teste de carga, teste de stress, teste de confiabilidade e teste de recuperação.

Além da derivação dos casos de teste, a rastreabilidade de testes, que segundo Gotel e Finkelstein (1995, tradução nossa) é a habilidade de descrever e acompanhar um requisito ao longo de um ciclo de vida, tanto na direção das suas fontes (pessoas, ideias, artefatos) como na direção dos artefatos que o sucedem é indispensável para obter um software com qualidade (CLELAND-HUANG, 2003; GOTEL; FINKELSTEIN, 1995; RAMESH; JARKE, 2001).

Sendo assim, esta pesquisa propõe a criação de uma técnica para priorizar os casos de testes feitos no sistema. Considerando-se uma tela de login, pode-se perceber que existem muitas variações de testes para serem executados para garantir que serão atendidos 100% dos requisitos dessa tela. Porém, isso seria inviável, devido à quantidade de tempo que seria consumido. Sendo assim, objetiva-se reduzir o tempo total gasto para executar os casos de testes, garantindo a qualidade do software.

2 Justificativa

Uma falha é um imprevisto que acontece quando as funcionalidades de um software não condizem com o objetivo real que ele deveria exercer (AVIZIENIS et al, 2004, tradução nossa).

O mesmo autor ainda afirma que um erro é a situação em que as funcionalidades de um sistema não equivalem à definição funcional do produto ou quando a definição não estabelece o objetivo do software de modo correto.

As falhas que surgem ao longo da utilização de um software, normalmente são acarretadas pela implementação incorreta da funcionalidade ou falha na sua definição. Softwares com este problema geram prejuízos como refazer o trabalho, extravio de dados, atrasos em operações relevante, entre outros.

Yourdon (1990) assegura que nenhum sistema é absolutamente protegido e isento de falhas. Isso ocorre, pois muitas funcionalidades não são muito utilizadas e consequentemente certos erros de maneira nenhuma serão encontrados, ou seja, o sistema conserva-se indefinidamente com defeito, não permitindo o seu reparo.

A criação de uma técnica para priorizar os casos de testes feitos em um sistema deve ser feita sempre pensando na qualidade de software. O emprego de tal estudo permitirá a construção de um processo baseado na matriz de rastreabilidade de testes aplicado à priorização de casos de testes. Tendo como objetivo validar as funcionalidades, buscando a qualidade.

Além disso, por ser a área de atuação profissional do pesquisador, trará maiores conhecimentos em relação aos conceitos utilizados no processo de teste e em relação aos casos de teste criados, avaliados, priorizados e executados na empresa em que atua, fazendo com que cresça profissionalmente.

O tempo utilizado para execução dos casos de teste será um dos principais fatores que influenciarão na priorização dos casos no processo de teste, porém necessitará que seja assegurada a qualidade do software, ou seja, este não é o fator mais relevante, já que não se pode deixar de executar um teste crítico em função do fator tempo.

A criação da matriz de rastreabilidade fará com que sejam controlados os casos de testes executados no processo. Além disso, serão documentados todos os detalhes do planejamento de teste, como a informação se o caso encontrou alguma falha ou não, se o teste foi planejado, caso sim, se foi também executado, informações do testador responsável e quando foi realizada esta execução.

3 Construção De Um Processo De Priorização De Casos De Testes

O emprego de testes nos softwares é indispensável para a garantia da qualidade do mesmo, principalmente para softwares que passam por

correções e aperfeiçoamentos contínuos, que segundo Sommerville (2003), possuem uma tendência maior para apresentar falhas.

Esta situação torna indispensável uma análise com base na priorização dos casos de testes referentes aos pontos do sistema que historicamente revelam a presença de erros. Fundamentado nisso, o tempo reservado aos testes será priorizado nos pontos em que existe a predisposição deles ocorrerem, postergando os casos de teste com menor propensão para serem executados por último. Essa técnica visa conservar ou ampliar a garantia em relação à funcionalidade, pois o foco será designado para o local em que teoricamente ocorrerá a maioria das falhas, utilizando um tempo menor para a efetuação deste procedimento.

O processo de priorização de casos de teste foi realizado com base na matriz de rastreabilidade, sendo que a escolha dos casos de teste se sucedeu por intermédio de método de derivação não formal, fundamentado no conhecimento e experiência do especialista de teste.

O presente trabalho é a construção de uma técnica para priorizar os casos de teste na validação das funcionalidades do sistema, utilizando como apoio a matriz de rastreabilidade. Desta forma, objetiva-se a redução do tempo total gasto para executar os casos de teste, porém garantindo a qualidade do software, visto que serão priorizados os casos de teste que historicamente apresentam erros. Para isso faz-se necessário a elaboração de uma metodologia com etapas bem divididas.

3.1 Estudo e seleção do software para realizar a execução dos casos de teste

O software desktop escolhido para a execução dos casos de teste foi o IRPF2015 - Declaração de Ajuste Anual, Final de Espólio e Saída Definitiva do País da Receita Federal. Foram analisadas as telas de Identificação do Contribuinte e do Cadastro de Dependentes.

O Imposto de Renda de Pessoa Física, segundo a Receita Federal é um imposto federal que acomete todos os cidadãos que receberam rendimentos tributáveis acima de um valor estabelecido. Anualmente é exigido a esse contribuinte o fornecimento de informações através da Declaração de Ajuste Anual - DIRPJ, para que seja averiguado se o contribuinte terá débitos (pagamento de imposto) ou créditos (devolução de imposto). O imposto é

determinado a partir da renda do contribuinte. A alíquota oscila equivalentemente de acordo com a renda tributável. Caso a pessoa não possua renda maior que o valor estabelecido, será considerada isenta.

Este software oferece diversos benefícios aos cidadãos que usufruem desse sistema, como a facilidade para declarar, eliminação de despesas com impressão e consulta online, em tempo real, da restituição, principalmente se comparado aos anos anteriores quando a declaração era um formulário preenchido manualmente ou preenchido eletronicamente e entregue em um disquete, além disso, não era possível consultar a situação da declaração, visto que era necessário aguardar notificação da Receita Federal em casa.

Escolheu-se esse software, por ser um sistema que é disponibilizado a todos os contribuintes e sua liberação coincidiu com o início do desenvolvimento do trabalho proposto. É primordial que este software não apresente falha, para isso, torna-se importante a realização de testes utilizando critério de priorização das principais funcionalidades do sistema, sobretudo as que historicamente apresentam defeitos.

Com a definição do software a ser utilizado no trabalho proposto, será necessário o aprofundamento dos conceitos das técnicas de derivação funcional, para proceder com o levantamento dos casos de teste.

3.2 Criação dos casos de teste de acordo com as três técnicas de derivação mais utilizadas

Os estudos destacaram que os métodos de derivação fundamentados em particionamento de equivalências, análise do valor limite, grafo causa-efeito, matriz ortogonal e experiência e conhecimento do especialista de teste podem ser aplicados de maneira suplementar. Para isso, torna-se essencial a aplicação de dois ou mais métodos de derivação conjuntamente para proporcionar casos de teste que alcancem maior produtividade para obter maior qualidade no software.

Foram escolhidos três dos métodos de derivação existentes - particionamento de equivalências, análise do valor limite e experiência e conhecimento do especialista de teste - para a criação dos casos de teste com base em seus conceitos.

É relevante considerar que conforme Rios e Moreira (2006), o método fundamentado na experiência e conhecimento do especialista de teste, pode ser desenvolvido com a pretensão de alcançar a máxima qualidade dos testes e do software.

Diante disso e por ser o método empregado na empresa em que o pesquisador trabalha foram escolhidos os casos de teste criados a partir dos conceitos da derivação fundamentada na experiência e conhecimento do especialista de teste, tendo como objetivo alcançar resultados mais efetivos.

A partir da criação dos casos de teste será realizada a montagem da matriz de rastreabilidade para cada ciclo de teste.

3.3 Criação da matriz de rastreabilidade

A matriz de rastreabilidade (TRM) armazena de forma íntegra as informações, levantadas dos dados históricos, dos casos de teste. Esse armazenamento de outro modo é relacionado a um ciclo de teste estabelecido.

Utilizando o software Microsoft Excel 2010 foram montadas sete tabelas, sendo uma planilha principal que contém a tabela dos ciclos de testes realizados e outras seis planilhas contendo as TRMs de casos de teste.

O requisito funcional escolhido para elaborar cada ciclo foi o sistema operacional utilizado na estação de trabalho. Este primeiro documento da matriz de rastreabilidade possui um link para acesso direto às planilhas de TRM de casos de teste. Baseado nessa planilha será possível averiguar os casos de testes planejados e executados.

Com a criação da matriz de rastreabilidade será imprescindível a realização de uma classificação para determinar o grau crítico de cada caso de teste, resultando assim na utilização da matriz de priorização GUT.

3.4 Classificação dos casos de teste com a matriz de priorização GUT

A priorização torna-se vital para inteirar-se por qual ponto começar o teste, servindo como uma classificação, de modo que primeiramente são efetuados os testes com maior criticidade até alcançar os menos críticos.

Para a realização do trabalho foi empregado o método da matriz de priorização GUT, que pontua cada caso de teste, com base em uma hipotética ocorrência de adversidade a partir desse caso de teste. São imputadas três notas de 1 a 5, em conformidade com as regras apresentadas, anteriormente na figura 8, para cada um dos parâmetros. Primeiramente é analisada a

Gravidade (impactos caso o problema não seja corrigido), posteriormente a urgência (prazo para correção) e por último a tendência (propagação do problema).

Com a conclusão do processo de pontuação dos casos de teste são multiplicadas as três pontuações gerando um valor de grau crítico, permitindo assim a classificação e organização da execução dos casos de teste.

A execução da pontuação dos casos foi analisada e efetuada pelo próprio pesquisador, que possui atuação profissional na área de teste, levantando no final o grau crítico de cada caso de teste.

Para a realização da priorização através da matriz de GUT foi empregado o princípio de Pareto conjuntamente a este método, com o objetivo de atingir uma melhor avaliação.

3.5 Priorização dos casos utilizando Pareto

O princípio de Pareto tornou-se uma ferramenta muito importante em diversos setores em que se necessita tomar decisões (FERNANDES, 1984), como no caso do trabalho proposto.

No presente trabalho, além de aplicar o método de priorização GUT, será empregado, conjuntamente, o princípio de Pareto com o propósito de alcançar um resultado superior. Para isso serão consideradas as notas obtidas com a aplicação da matriz de priorização GUT e será planejada a execução de 20% dos casos de teste levantados para cada ciclo, sendo que serão priorizados os que possuem maior grau crítico, resultado da multiplicação entre os parâmetros Gravidade, Urgência e Tendência.

Para os ciclos de testes que possuem os mesmos casos de teste levantados, serão desconsiderados os escolhidos nos ciclos anteriores. Além disso, deve-se considerar que o cálculo da escolha dos 20% dos casos deverá desconsiderar os casos de teste já escolhidos nos ciclos anteriores, diminuindo a quantidade de casos de teste de um ciclo para outro.

Com a priorização e escolha dos casos de testes concluída, sucede-se a execução dos casos para verificar o comportamento obtido e descrever as informações necessárias a cada item.

3.5 Execução dos casos de teste

Após a seleção dos casos de teste foi verificado que os casos com menor prioridade ficaram de fora dos ciclos de teste, dando a entender que não há a necessidade de executá-los.

Todavia, para os casos de teste planejados, a execução foi realizada. Para os testes que apresentaram resultado positivo, a coluna *Passou* da planilha foi preenchida com um *x*. Em contrapartida, foi preenchida a coluna *Falhou* para os que não resultaram em um comportamento esperado, ou seja, apresentaram adversidades em relação ao que foi anotado na descrição do caso de teste. Além disso, deve-se registrar a adversidade ocorrida em uma ferramenta que reporte as falhas, preenchendo o *Issue* (Código do erro) em conjunto com as colunas *Data*, *Quem testou* e o tempo consumido para a execução do caso de teste.

A etapa de explanação da metodologia chega ao final com a execução dos casos de teste. Concluindo, assim, a criação do método de priorização dos casos de teste.

4 Resultados Obtidos

O estudo atingiu os objetivos desejados em relação à criação de um método de priorização dos casos de teste.

A figura 1 contém o processo validado, expondo os percentuais dos casos de testes planejados, executados, os que possuíram resultado positivo e os que apresentaram falhas.

Figura 1 – Processo validado – Software IRPF

Processo validado						
	TRM 1.0	**TRM 2.0**	**TRM 3.0**	**TRM 4.0**	**TRM 5.0**	**TRM 6.0**
Quantidade planejada	20%	20%	20%	20%	20%	20%
Quantidade executada	20%	20%	20%	20%	20%	20%
Quantidade "Passou"	100%	100%	100%	100%	100%	100%
Quantidade "Falhou"	0%	0%	0%	0%	0%	0%

Fonte: Do autor.

Os percentuais em relação à *Quantidade planejada* e a *Quantidade executada* possuíram os valores conforme a métrica apresentada na matriz de priorização GUT associada ao Princípio de Pareto e a técnica de derivação de conhecimento e experiência do especialista de teste.

No resultado da execução, *Quantidade "Passou"* e *Quantidade "Falhou"*, pode-se perceber que não foram encontradas falhas, neste caso, considera-se que o software de Declaração do Imposto de Renda de Pessoa Física da Receita Federal é um sistema estável, que já passou por diversas baterias de testes antes de ser liberado ao usuário, condição imprescindível para um programa desse porte. Seria, portanto, inviável disponibilizá-lo apresentando erros, visto que uma falha poderia acarretar em atrasos na entrega do imposto pelo contribuinte ou, ainda, permitir seu envio com o preenchimento incorreto.

Foi constatado que o software, da Receita Federal, escolhido no início do desenvolvimento do trabalho proposto possui uma grande estabilidade, já que são implementadas poucas melhorias de uma versão para outra, permitindo assim que seja dada atenção aos problemas existentes e consequentemente realizando sua correção.

Portanto, para proceder com a validação da criação da técnica para priorização dos casos de teste foi necessário escolher outro software que não fosse estável como o software da Receita Federal. Foi realizada pesquisa no site da Anvisa, procurando um serviço que fosse disponibilizado a qualquer usuário e escolheu-se o Formulário para denúncias, reclamações e solicitações[4].

Neste software, também foram realizados todos os passos, desde o levantamento dos casos de teste utilizando a técnica de derivação, passando pela classificação utilizando GUT e pela priorização utilizando o Princípio de Pareto até a execução dos casos de teste.

A figura 2 compreende o processo validado, apresentando os percentuais dos casos de testes planejados, executados, os que possuíram resultado positivo e os que apresentaram falhas.

[4] Disponível em <http://www10.anvisa.gov.br/ouvidoria/CadastroProcedimentoInternetACT.do?metodo=inicia>. Acesso em 31 mai. 2015.

Figura 2 – Processo validado – Software Formulário para denúncias, reclamações e solicitações

Processo validado						
	TRM 1.0	TRM 2.0	TRM 3.0	TRM 4.0	TRM 5.0	TRM 6.0
Quantidade planejada	20%	20%	20%	20%	20%	20%
Quantidade executada	20%	20%	20%	20%	20%	20%
Quantidade "Passou"	25%	100%	40%	25%	100%	100%
Quantidade "Falhou"	75%	0%	60%	75%	0%	0%

Fonte: Do autor.

Com o processo validado após a aplicação dos testes no segundo software, pode-se constatar que os objetivos propostos no inicio da pesquisa foram alcançados, proporcionando como resultado final uma técnica de priorização dos casos de teste que continua garantindo a qualidade do sistema, porém permitindo a redução do tempo consumido para a realização dessa atividade.

5 Conclusão

As empresas sempre se empenham em fornecer um produto de qualidade para seus consumidores, no caso de uma empresa de software isso não é diferente. O item principal para estabelecer a qualidade de um sistema é o teste de software e é por meio dele que serão analisados os comportamentos do sistema, para avaliar se realmente foram alcançados os resultados esperados.

Comprovou-se que a utilização de métodos de priorização dos casos de teste proporciona um aproveitamento superior do tempo destinado para esta etapa, uma vez que se almeja optar pela execução dos casos de teste em pontos do sistema, que historicamente, apresentam adversidades.

Com esse objetivo que foi elaborado o processo de priorização de casos de testes, a fim de validar as funcionalidades dos softwares. Assim, antes de iniciar a execução dos testes, é possível classificar os casos de testes que possuem maior propensão a detectar erros, podendo priorizá-los.

Para alcançar o estabelecido nesse trabalho, foi adquirido conhecimento na área de teste de software, principalmente, em relação aos conceitos e técnicas. Utilizando o critério de derivação baseado no conhecimento e experiência do especialista foram levantados os casos de teste. Posteriormente, o foco foi concentrado na classificação dos casos de

teste utilizando a matriz de priorização GUT e planejando os casos a serem executados a partir da priorização, utilizando o princípio 80/20 de Pareto.

Como resultado final, foi criado um processo baseado na matriz de rastreabilidade aplicado à técnica de priorização de casos de teste, que foram registradas em planilha eletrônica, na qual os casos de teste são detalhados, possibilitando sua utilização em futuras consultas. Desta maneira, as informações documentadas servirão como um histórico para os testadores. Posto isto, permite-se concluir que o trabalho conseguiu alcançar seu objetivo geral.

Os estudos realizados neste trabalho proporcionaram um ganho profissional ao pesquisador, visto que obteve compreensão dos conceitos das técnicas que utiliza no dia-a-dia, podendo praticá-las assimilando seu funcionamento também na teoria. Além disso, teve a oportunidade de agregar esses métodos na sua atividade profissional, apresentando melhoria na qualidade dos seus testes e otimizando o seu tempo de forma mais satisfatória.

Dando continuidade a esta pesquisa, tem-se como possibilidade de trabalhos futuros a:

a) criação de outra técnica de priorização dos casos de teste para fazer comparação ao criado;
b) implementação dessa metodologia em Java;
c) ampliação das variáveis de ambiente utilizadas na TRM;
d) aumento da cobertura dos requisitos funcionais e, principalmente, dos requisitos não funcionais;
e) implantação dessa metodologia em uma empresa de software, garantindo uma validação ainda mais confiável.

6 Referências

AVIZIENIS, A. et al**.** Basic Concepts and Taxonomy of Dependable and Secure Computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n.1, p. 11–33, 2004. Disponível em:<http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1335465>. Acesso em 22 nov. 2014.

DIAS NETO, Arilo Claudio. **Introdução a Teste de Software**. Qualidade de Software: Entenda os principais conceitos sobre Testes e Inspeção de Software. São Paulo, 2007.

FERNANDES, José Carlos de F. Administração de material um enfoque sistêmico. 2 ed. Rio de Janeiro: Livros Técnicos e Científicos, 1984.

GOTEL, O. e FINKELSTEIN, A. (1995) "**Contribution Structures**", Proceedings of 2nd Internationak Symposium on Requeriments Engineering. IEEE Computer Society Press.

PRESSMAN, R. S. **Engenharia de Software.** 6 ed. São Paulo: McGraw-Hill, 2006.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software:** Segunda edição revisada e ampliada. 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.

YOURDON, Edward. **Análise estruturada moderna.** 10. ed. Rio de Janeiro: Campus, 1990. 836p.