UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

CLEVERSON REINERT

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

CRICIÚMA

2012

CLEVERSON REINERT

UTILIZAÇÃO DE TESTES AUTOMATIZADOS PARA AMPLIAR A COBERTURA FUNCIONAL NA MANUTENÇÃO 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

2012

CLEVERSON REINERT

UTILIZAÇÃO DE TESTES AUTOMATIZADOS PARA AMPLIAR A COBERTURA FUNCIONAL NA MANUTENÇÃO 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, 26 de novembro de 2012

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

AGRADECIMENTOS

Agradeço inicialmente a Deus por alimentar meu espírito com motivação nos momentos mais extremos. Igualmente agradeço à minha família, em especial a meus pais Beto e Sinha por mostrarem a importância transformadora do conhecimento, assim como aos meus irmãos Edson e Robson que entenderam minhas ausências, principalmente nas pescarias em que não pude ir. Deixo um agradecimento especial à minha esposa Andréia, seus pais José e Reni e aos meus cunhados Tiago, Tamara, Daniela e Cristobal, que procuraram compreender minhas ausências e foram meu esteio em vários momentos. Obrigado a todos vocês pelos anos de amor, dedicação e aconselhamentos. Também agradeço aos professores de Ciência da Computação da UNESC por transmitirem seus conhecimentos e experiências, em especial à professora Merisandra Cortês de Mattos pelo apoio, conselhos e cobrança para "não deixar a peteca cair". Ao professor Gustavo Bisognin, meu orientador, agradeço pela sabedoria, conselhos e experiências profissionais compartilhadas de forma generosa, e que foram cruciais para a conclusão do trabalho. Agradeço de forma geral aos amigos e colegas de trabalho por compreender as ausências necessárias frente ao desafio. Aos amigos Adriano Dias, Ferreira, Almeida, Antonio Marcos, Birolo, Carla, Flaris, Jaisson, Jeferson, Henrique, Marcelo Pra, Michael, Reginaldo Darolt, Roberto, Sergio Pochmman de Palotina e Vinicius Tessele de Palotina agradeço pelas palavras que me permitiram avaliar as oportunidades por trás das dificuldades. Também agradeço ao Sensei Neucir Cardoso de Curitiba, por seu exemplo de caráter, persistência e respeito, determinantes para o trabalho e para minha vida. Agradeço especialmente ao Marcondes, meu diretor técnico, mentor profissional e amigo pelos anos de confiança e incentivo, bem como por permitir o uso do software Domínio Atendimento nesse trabalho. Da mesma forma agradeço à empresa Domínio Sistemas pelos desafios que me permitiram aplicar diversos conhecimentos ao longo dos anos. Agora agradeço especialmente à minha equipe de testes de software e à minha ex-equipe de Suporte às Unidades da Domínio. A liderança de tais equipes me presenteou com experiências que me permitem dizer que todos vocês são exemplos de pessoas que superam seus desafios pessoais e profissionais. Por isso vocês foram e são pessoas importantes para a minha vida e para esse trabalho. Por fim e não menos importante, agradeço aos meus examinadores Gustavo, Ana e Gilberto pelas palavras generosas e por exporem de forma precisa os pontos a melhorar no meu trabalho.

“Se eu tivesse oito horas para derrubar uma árvore, passaria seis afiando o meu machado”

Abraham Lincoln

RESUMO

A qualidade dos produtos de software se consolidou como uma necessidade. Com isso, profissionais da engenharia de software buscam a evolução de processos e produtos. Condizente com esse contexto, o presente trabalho visa apresentar o uso da automação de testes funcionais regressivos como uma alternativa para ampliar a cobertura funcional em testes de softwares que sofrem constantes manutenções. O trabalho inicia com estudos sobre conceitos da engenharia de software, como por exemplo, a qualidade de software, seus modelos, métodos e normatizações, bem como teorias voltadas à manutenção e manutenabilidade do software. Na sequência os testes funcionais são avaliados, explorando as técnicas de teste caixa-preta e caixa-branca, com aprofundamento nas fases, tipos de testes e critérios de derivação do teste caixa-preta. Nesse momento a fase de testes de regressão também recebe estudos que demonstram a sua importância para validar requisitos funcionais após uma manutenção de software. Após isso o trabalho aborda os conceitos da automação de testes, suas ferramentas, riscos de implantação e fatores de sucesso. No âmbito da prática, empregando como critério de derivação o conhecimento e experiência do especialista, o trabalho contempla a criação de uma base de testes e uma matriz de planejamento dos testes, esta última contendo uma lista de requisitos funcionais que facilitam a priorização e seleção de casos de testes condizentes com determinada manutenção. Na etapa seguinte ocorre a seleção da ferramenta Selenium IDE para gravar e executar os scripts de automação, assim como a escolha do software Domínio Atendimento como a solução a ser testada. Nessa fase os casos de testes são selecionados na matriz de planejamento dos testes e submetidos à execução automatizada, com coleta dos tempos de execução dos testes automatizados e manuais. Finalmente, com base em hipóteses de manutenções levantadas para fins didáticos, os resultados são avaliados e as comparações permitem definir se o objetivo geral do trabalho foi atingido.

Palavras-chave: ENGENHARIA DE SOFTWARE. AUTOMAÇÃO DE TESTES FUNCIONAIS. TESTE DE REGRESSÃO. COBERTURA FUNCIONAL.

ABSTRACT

The quality of software products has established itself as a necessity. With this professional software engineering seek the development of processes and products. Befitting this context, this work presents the use of automated functional testing as an alternative to extend the regressive functional test coverage in software testing that suffer constant maintenance. The work begins with studies on concepts of software engineering, such as software quality, its models, methods and norms, as well as theories focused on maintenance and maintainability of the software. Following the functional tests are evaluated by exploiting the techniques of testing black-box and white-box, with further phases, types of tests and criteria derivation of black-box testing. In this moment the phase of regression testing also receives studies that demonstrate its importance to validate the functional requirements after a software maintenance. After that, the work discusses the concepts of test automation, their tools, deployment risks and success factors. Under the practice, employed as a criterion for derivation based in the specialist's knowledge and experience, the work includes the creation of a database of tests and test planning matrix, the latter containing a list of functional requirements that facilitate the prioritization and selection test cases consistent with certain maintenance. In the next step is the selection tool Selenium IDE to write and run scripts for automation, as well as the choice of software Domínio Atendimento as the solution to be tested. In this phase test cases are selected in the planning matrix of tests and submitted to automated execution, with runtimes collection of automated and manual tests. Finally, based on hypotheses of maintenance raised for didactic purposes, the results are evaluated and comparisons allow to set the overall goal of the study was achieved.

Keywords: SOFTWARE ENGINEERING. FUNCTIONAL TESTING AUTOMATION. REGRESSION TESTING. FUNCTIONAL COVERAGE.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

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

SUMÁRIO

1 INTRODUÇÃO

A busca pela qualidade no desenvolvimento de software tem recebido uma crescente atenção de entidades públicas e privadas atuantes no setor (GUERRA; COLOMBO, 2009; TENÓRIO JUNIOR, 2009). Inserida neste contexto, a engenharia de software tem evoluído e contribuído desde 1968, quando os primeiros estudos sobre a construção de sistemas já apontavam limitações que ampliavam os custos de produção e manutenção de software (SOMMERVILLE, 2003).

Atualmente as companhias se valem dos avanços tecnológicos para desenvolverem melhores produtos e serviços, de implementação rápida e com custos menores, porém a tecnologia também impõe desafios mais complexos ao requerer uma integração dos softwares a sistemas e componentes de terceiros, em alguns casos empregados no processo produtivo (CHRISSIS; KONRAD; SHRUM, 2007, tradução nossa). Outros desafios da engenharia de software são os sistemas legados, a heterogeneidade de ambientes ou componentes, bem como o fornecimento do produto em um tempo hábil, respeitando prazos contratuais (SOMMERVILLE, 2003).

Conforme Pressman (2006), o software possui uma linha evolutiva e apresentará defeitos durante a existência, alguns devido à evolução, pois quanto maior a complexidade, aliada a manutenções constantes, maiores serão as falhas e a queda na qualidade do software.

Essa preocupação com a qualidade no desenvolvimento de software é intensa a tal ponto que políticas públicas já foram adotadas, como a recente criação do Programa Brasileiro de Qualidade e Produtividade em Software (PBQP), elaborado para elevar a capacidade competitiva e produtiva do setor, pois incentiva o estudo e aplicação das melhores práticas para o desenvolvimento de sistemas, com uso de ferramentas, normatizações, metodologias e técnicas que visam qualidade (BRASIL, 2006; GUERRA; COLOMBO, 2009).

Condizentes com esta preocupação, os métodos de desenvolvimento evoluíram visando prevenir e eliminar defeitos no produto, porém é sabido que as falhas ainda ocorrem, produzindo queda na qualidade do software, o que torna imprescindível o emprego de técnicas que cooperem para a detecção das falhas, como por exemplo, a adoção de equipes de testes na tentativa de garantir a qualidade desejada, antes que o produto se torne operacional (BURNSTEIN, 2010, tradução nossa; DELAMARO; MALDONADO; JINO, 2007).

Segundo Viccari (2009), o teste de software é uma tarefa bastante difundida como garantia de requisitos e de qualidade, porém são rotinas basicamente manuais e repetitivas. Apesar de o ser humano ter a capacidade cognitiva que lhe dá vantagem na seleção das melhores variações em testes, diante da repetição de tarefas simples, como por exemplo, a produção massiva de casos de testes, os humanos demoram na execução e são propensos a gerar erros (PEZZÈ; YOUNG, 2008).

Frente a estes aspectos, o gerenciamento do processo de testes requer o uso da automação de testes de software, visando minimizar o tempo e custo de produção, com ampliação da qualidade final do produto. Todavia, a automação de testes também tem seus contratempos, tendo em vista que é um investimento de longo prazo, que engloba reestruturação de equipes, reorganização de processos e em alguns casos a aquisição e implantação de ferramentas. A produção de alguns impactos também precisa ser avaliada de forma aproximada, como por exemplo, um possível aumento no tempo de manutenção (*scripts*) e no tempo de preparação de testes (RIOS; MOREIRA, 2006).

Diante dos pontos apresentados, esta pesquisa propõe a utilização de testes automatizados na cobertura das funcionalidades do software, empregando-se critérios formais ou não formais para derivação dos casos de testes, como por exemplo, particionamento de equivalências, teste de matriz ortogonal ou o próprio conhecimento e experiência do especialista de testes, entre outros. Com isso, objetiva-se 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.

1.1 OBJETIVO GERAL

Demonstrar a utilização de testes automatizados para ampliação da cobertura funcional em softwares de manutenção constante.

1.2 OBJETIVOS ESPECÍFICOS

O presente trabalho aborda temas referentes aos seguintes objetivos específicos:

a) conhecer os conceitos de qualidade na Engenharia de Software;
b) entender o uso da tarefa de testes na manutenção de software;
c) empregar critério para derivação dos casos de testes;

d) aplicar testes automatizados para cobertura funcional;
e) analisar os resultados da aplicação dos testes automatizados para cobertura funcional.

1.3 JUSTIFICATIVA

Os sistemas computacionais tiveram grande evolução nos últimos 50 anos, saindo da categoria de programas especialistas e científicos, para produtos de software comerciais. Nesse âmbito a engenharia de software acompanhou essa evolução sempre preocupada com a qualidade dos sistemas desenvolvidos (PRESSMAN, 2006).

O emprego de softwares se tornou tão comum ao ponto que sua existência só é percebida quando ocorre uma falha, por exemplo, no uso de um caixa automático que apresente a frase "Falha geral no sistema, tente utilizar outro caixa." ou quando uma reportagem cita um software como possível causa de um acidente aéreo. Apesar do nível crítico bem distinto, ambos os exemplos remetem à baixa qualidade em softwares.

Estes exemplos demonstram o quão importante é o estudo da engenharia de software, a qual, segundo Sommerville (2003), basicamente possui ferramentas e métodos voltados à manutenção prática e à obtenção da confiabilidade do produto de software. Essa visão simplificada parece limitar o raio de ação da engenharia de software, contudo uma avaliação criteriosa demonstra que a aplicação da abordagem de engenharia para a produção de software emprega princípios científicos, métodos, modelos, padrões e teorias que possibilitam gerenciar, planejar, modelar, desenhar, implementar, mensurar, analisar, manter e evoluir sistemas de software, tudo isso com a produção econômica de softwares qualificados (PETERS; PEDRYCZ, 2000, tradução nossa).

Indiferente ao grau crítico de uma falha de software, detectar, identificar, definir ações para solução da mesma e dar manutenção tentando evitar que novos erros surjam, constituem tarefas árduas, porém necessárias (SOMMERVILLE, 2003). Segundo apontam Pezzè e Young (2008), tanto um software quanto um carro de corrida são produtos únicos, de comportamentos variáveis, pois poderão ser submetidos a circunstâncias diversas, exigindo que o teste e a avaliação da qualidade seja algo exclusivo para cada novo exemplar do produto.

É necessário considerar ainda que as falhas de software estão presentes desde o início do projeto, devido a isso a fase de testes compreende diversas etapas ou métodos que

visam identificar erros inerentes ao processo produtivo, desde falhas no desenvolvimento de classes, métodos ou funções, até possíveis imperfeições em requisitos do software (DELAMARO; MALDONADO; JINO, 2007). Para este fim a engenharia de software prevê técnicas como o teste de componentes ou de unidade, teste de integração e teste de sistemas ou de defeitos. Os testes de sistemas ou de defeitos são divididos entre teste caixa-branca e teste caixa-preta, este último também chamado de teste funcional (DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006; SOMMERVILLE, 2003).

A tarefa de testes é uma rotina que permite a combinação de várias técnicas visando a qualidade, contudo a técnica dos testes funcionais será um dos focos do trabalho atual, justamente porque segundo Sommerville (2003) esta permite testes bastante abrangentes quanto aos requisitos funcionais do software, além desta técnica possibilitar o uso de vários critérios para derivação dos casos de testes, dentre os quais pode-se citar os critérios de teste baseado em grafos, particionamento de equivalências, análise de valor limite e teste de matriz ortogonal (PRESSMAN, 2006). É possível acrescentar a esta lista os critérios de teste funcional sistemático e de *error* guessing (DELAMARO; MALDONADO; JINO, 2007), bem como o critério não formal baseado na experiência e conhecimento do especialista de testes (RIOS; MOREIRA, 2006).

A abordagem sistemática com uso de critérios, de acordo com Pezzé e Young (2008), é a alternativa ideal para derivar casos de testes, quando comparada ao uso de força bruta, pois reduz os esforços da equipe de testes e possibilita maior abrangência na cobertura funcional. Ainda assim, os testes são uma etapa de alto custo e que consome grande tempo do projeto, em alguns casos chegando a 50% do tempo total de entrega do produto de software, fatores estes que necessariamente exigem o uso de ferramentas de apoio, como os aplicativos para automação de testes (SOMMERVILLE, 2003).

O emprego da automação de testes se mostra importante por ser um recurso que reduz de forma incomensurável o tempo de teste e amplia a qualidade do produto, pois refina a exatidão do software frente aos requisitos funcionais (PEZZÈ; YOUNG, 2008).

Frente aos fatores expostos, considerando-se o cenário de crescente busca pela qualidade sem prejuízo aos prazos de projetos, o trabalho atual mostra-se relevante, já que os testes automatizados apresentam-se viáveis para ampliar a cobertura funcional, elevando a qualidade desejada e reduzindo o tempo final do projeto de software. Vale considerar que regionalmente são poucos os trabalhos acadêmicos similares que enfocam rotinas automatizadas de testes funcionais, o que foi observado durante a fase do levantamento

bibliográfico e enseja a abertura de mais opções para trabalhos futuros nessa área da engenharia de software.

1.4 ESTRUTURA DO TRABALHO

Os temas abordados pelo trabalho foram distribuídos em 7 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, A Engenharia de Software:** trata de conceitos ligados à engenharia de software, relatando a importância crescente dos softwares e a necessidade de se profissionalizar as pessoas atuantes no setor. Também foram apresentados os ganhos proporcionados pela engenharia quanto à qualidade do software e a qualidade do próprio processo produtivo. Nesse âmbito foram tratados os modelos genéricos de produção de software, bem como modelos, normatizações e métricas de qualidade desenvolvidos por entidades nacionais e internacionais. Foram introduzidas ideias iniciais sobre a importância de se aplicar os testes de software na busca da qualidade dos produtos. Nos subcapítulos foram abordados temas que passaram pelos conceitos de qualidade de software, modelos de qualidade, os impactos da manutenção de software juntamente com a importância da manutenabilidade para a qualidade do software;

b) **capítulo 3, Testes de Software:** comporta temas ligados ao uso de testes de software na validação da qualidade, introduzindo conceitos sobre validação ou invalidação de qualidade, demonstrando as limitações da implantação do processo de testes, assim como as vantagens da sua adoção. Também são introduzidos conceitos de testes de software, entre eles, as técnicas de testes, tipos de testes e suas fases de aplicação. Definições específicas, como por exemplo, critérios de derivação e suas variações também são abordadas no capítulo. Nos subcapítulos foi enfatizada a técnica de teste caixa-preta e seus critérios de derivação. Os conceitos da técnica caixa-branca também foram expostos, bem como os tipos de testes, para as duas técnicas. Os subcapítulos

abordaram ainda o ciclo de vida dos testes de software, além de serem detalhadas as fases de testes da técnica caixa-preta;

c) **capítulo 4, Automação de Testes Funcionais:** o capítulo aborda desde conceitos iniciais sobre a prática de testes automatizados, quanto o seu uso especificamente com testes funcionais. Também são apresentadas informações que validam a importância do uso de testes automatizados. Alguns cuidados importantes quanto à adoção de testes automatizados também são expostos, passando pelos tipos de softwares de automação de testes, dando detalhes relevantes de cada um. Nos subcapítulos são detalhadas as limitações e os fatores de sucesso na adoção do processo de automação;

d) **capítulo 5, Ferramentas para Automação de Testes Funcionais:** esse capítulo apresenta algumas das ferramentas disponíveis para a prática da automação de testes por meio da gravação e execução de *scripts* de testes. Os softwares expostos foram o Selenium IDE (SELENIUM, 2012, tradução nossa), Autoit (AUTOIT, 2012, tradução nossa) e Testcomplete (SMARTBEAR, 2012, tradução nossa). No subcapítulo foram detalhadas as características e funcionalidades do Selenium IDE;

e) **capítulo 6, Trabalhos Correlatos:** o capítulo faz uma referência introdutória sobre os objetivos de se estudar os trabalhos correlatos. Os subcapítulos detalham o resultado dos estudos realizados nos 4 trabalhos correlatos levantados na fase da pesquisa bibliográfica, os quais têm os seguintes títulos,

   - Desenvolvimento de Uma Metodologia Aplicada ao Gerenciamento e Acompanhamento de Testes de Software via Web (LOPES, 2009),

   - Autotec: Uma Ferramenta de Automação de Testes para Interfaces Dinâmicas (MÜNCHEN, 2010),

   - Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo (CRESPO et al, 2012),

   - Teste de Software Automatizado: Uma Solução para Maximizar a Cobertura do Plano de Teste e Aumentar a Confiabilidade e Qualidade em Uso do Software (CATELANI et al, 2010, tradução nossa);

f) **capítulo 7, Automação de Testes Regressivos na Ampliação da Cobertura Funcional de Softwares com Manutenções Frequentes:** trata do trabalho desenvolvido, introduzindo conceitos que levaram à realização do trabalho e

apresentando a metodologia aplicada. O último subcapítulo retrata as práticas, estudos e resultados que o trabalho produziu frente aos objetivos desejados. Nos demais subcapítulos são detalhadas as etapas que compuseram a metodologia, que são,

- Levantamento Bibliográfico,
- Estudo Sobre as Técnicas, Ferramentas e Modelos de Qualidade da Engenharia de Software,
- Estudo Sobre a Tarefa de Teste Caixa-preta,
- Estudo e Seleção do Critério de Derivação dos Casos de Testes,
- Estudo e Seleção da Ferramenta de Automação dos Testes Funcionais,
- Estudo e Seleção do Software *Web* a Testar,
- Composição da Base de Testes,
- Composição da Matriz de Planejamento dos Testes,
- Execução dos Testes Manuais com Gravação da Automação e Criação dos *Scripts*,
- Execução da Automação de Testes.

2 A ENGENHARIA DE SOFTWARE

O software é um produto intangível, abstrato e maleável (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa; PRESSMAN, 2006), pois é inerte a leis físicas ou a padrões da industrialização, o que reduz os limites para a produção, mas também a torna mais intrínseca (SOMMERVILLE; 2003).

Clientes finais veem o software como um produto que provê recursos, simplifica rotinas do cotidiano, processa dados, produz informações e agrega conhecimento, enquanto que na visão de desenvolvedores é um sistema composto por programas, bibliotecas, suplementos, requisitos funcionais e documentações (PRESSMAN, 2006).

Essas visões e características convergem para a definição de que o software é um produto imaterial e flexível, elaborado em um crítico processo produtivo, que une a visão e conhecimento técnico do desenvolvedor às necessidades funcionais objetivadas pelo cliente final. Segundo Burnstein (2010, tradução nossa) e Sommerville (2003), no âmbito da qualidade, o software ainda precisa comportar quatro atributos básicos, que são a manutenabilidade, confiabilidade, eficiência e praticidade.

Atualmente o software é uma das bases dos sistemas de controle financeiro, navegação aérea, comunicação, saúde à distância, segurança nacional, entre outras áreas, com importância estratégica para a política e a economia global, o que exige a profissionalização de quem projeta, desenvolve e evolui tal produto. Conforme Pressman (2006) aponta, o software deve ser tratado cuidadosamente, considerando que dele dependem vidas, negócios e até a continuidade de governos.

Partindo dessa ótica, a abordagem de engenharia aplicada à produção de software, tem a importante missão de agregar qualidade, produtividade e redução de custos (PETERS; PEDRYCZ, 2000, tradução nossa; SOMMERVILLE, 2003).

A engenharia de software materializa a evolução do produto e do próprio processo produtivo ao herdar características de outras engenharias, adotando métodos sistematizados e reutilizáveis para a produção (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa). Nessa busca, a qualidade, a produtividade e a redução de custos são obtidas por meio de ações como o gerenciamento, planejamento, modelagem, desenho, implementação, mensuração, análise, manutenção e evolução dos sistemas de software, que ocorrem graças ao emprego de princípios científicos, métodos, modelos, padrões e teorias postuladas pelas engenharias

(PETERS; PEDRYCZ, 2000, tradução nossa), as quais podem ser aplicadas à produção de Software.

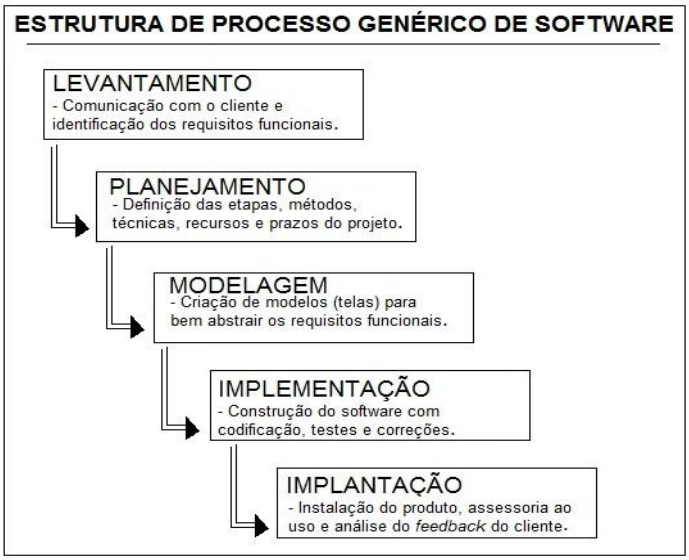
A aplicação desses recursos ocorre em uma sequência de estágios denominada de *software lifecycle* ou ciclo de vida do software (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa), no qual o processo de software evolui de uma etapa para outra com ações e tarefas encadeadas, que garantem que um estágio só inicie quando outro estiver completamente concluído (LEDGARD, 1987, tradução nossa; PRESSMAN, 2006), dando fluência e agregando dinamismo ao processo de produção.

Cada produto de software tem características próprias quanto ao rigor contra falhas, complexidade, tamanho do projeto ou urgência de entrega. Isso exige modelos distintos de produção que satisfaçam critérios necessários de qualidade e pontualidade para cada projeto.

Esses são os motivos pelos quais o processo de software comporta diversos modelos padronizados, como por exemplo, em cascata, incremental, espiral e evolucionário (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa; PETERS; PEDRYCZ, 2000, tradução nossa). Para Pressman (2006) entram nessa lista o modelo concorrente e os baseados em componentes, métodos formais, orientação a aspectos ou em processo ágil.

Apesar das variações quanto a sua aplicabilidade, os modelos padronizados têm diversas atividades em comum. Por conseguinte, ainda segundo Pressman (2006), a estrutura de processo genérico para a engenharia de software demonstrada na figura 1, abrange as atividades mais facilmente aplicáveis a diversos projetos.

Figura 1 – Estrutura de processo genérico de software.



Fonte: Adaptado de Pressman (2006).

Observando o aspecto humano para as atividades expostas na figura 1, é importante destacar que a própria natureza destas facilita a execução do processo ao exigir sinergia das equipes, o que denota outra faceta que a abordagem de engenharia traz, a de dar uma característica de time (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa) aos envolvidos no projeto de produto de software. O impacto positivo é que com isso são diminuídas as chances de ocorrências comuns envolvendo equipes diante de novos paradigmas, como por exemplo, a não aceitação de novas diretrizes, dificuldade em visualizar ou concordar com resultados almejados, não comprometimento com prazos ou com padrões de qualidade, entre outras.

Retornando ao âmbito técnico e tomando como referência a estrutura genérica de processo de software proposta por Pressman (2006), é necessário ressaltar que na tarefa de construção está inserido um dos principais focos do trabalho atual, a verificação do produto com base em testes funcionais do software. Essa técnica é importante na obtenção da qualidade do produto, uma vez que conforme Molinari (2010), o investimento em teste de software resumidamente é o mesmo que investir em qualidade.

Os estudos da engenharia de software englobando a adoção dos modelos, atividades e técnicas já citadas, proporcionaram avanços notórios na fluência dos projetos rumo à qualidade dos produtos, à pontualidade e redução de custos (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa; PETERS; PEDRYCZ, 2000, tradução nossa; PRESSMMAN, 2006; SOMMERVILLE, 2003). Contudo ainda se mostra essencial a medição do impacto positivo que cada etapa do clico de vida proporciona ao processo de software como um todo. Isso retroalimenta a cadeia de produção, pois permiti que a qualidade seja atingida e revigorada a cada iteração do ciclo produtivo.

A percepção dessa necessidade tem conduzido entidades internacionais a cooperar com a engenharia de software, se dedicando a pesquisas de normas e métodos que melhorem o processo e o produto de software. Tudo ocorre com base na mensuração e certificação de vários critérios de qualidade. Entre tais entidades merecem destaque a *International Organization for Standardization* (ISO), *American National Standards Institute*/*Institute of Electrical and Electronics Engineers* (ANSI/IEEE) (GUERRA; COLOMBO, 2009), *Software Engineering Institute* (SEI) (PRESSMAN, 2006; SEI, 2010, tradução nossa) e a Associação para Promoção da Excelência do Software Brasileiro (SOFTEX) (SOTEX, 2011), entre outras. Na lista das normatizações aplicáveis ao processo de software estão a NBR ISO/IEC 9126

(ABNT, 2003), *Capability Maturity Model Integration* (CMMI) (SEI, 2010, tradução nossa) e MPS.br (COUTO, 2007; SOFTEX, 2011).

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

2.1 QUALIDADE DE SOFTWARE

Em sua morfologia a palavra qualidade significa a propriedade, atributo ou condição que distingue um bem, ser ou serviço dos demais, dando-lhe características que numa escala de valores, permitem uma avaliação, podendo o avaliador aceitá-lo ou renegá-lo.

Um veículo movido a gasolina, por exemplo, pode ter baixa qualidade para alguém com consciência ecológica, enquanto que outro indivíduo focado no rendimento poderá indicar a qualidade como alta, indiferente do combustível.

Resumidamente a qualidade vai depender das particularidades do elemento avaliado versus os anseios de quem está avaliando o produto com base em determinadas características. Com isso o foco correto passa a ser algo essencial na busca da qualidade, exigindo que não só o produto seja conhecido, mas também seu público alvo, que deve ser amplamente estudado, seus anseios capturados, dissecados e convertidos em conhecimento a ser esculpido no produto. Critérios que definam um bom nível de satisfação devem ser estruturados e medidos. Por fim cada característica deve atingir aquilo que se convencionou como a qualidade ideal. Nessa lógica o resultado será um produto que, tendo um foco produtivo direcionado ao cliente, trará satisfação e aprovação quanto a sua qualidade.

Essa contextualização não pode ser aplicada de forma pura e simples para o software, pois ele é um produto imaterial e flexível (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa), um fruto do intelecto humano (SOMMERVILLE, 2003), tornando necessário um refinamento na definição da sua qualidade. Esse é um dos motivos que dificultam a determinação do que é qualidade de software. Teoricamente ela é conquistada ao serem contemplados os requisitos funcionais desejados pelo cliente. Na prática só isso não basta, é preciso obter características como a manutenabilidade, confiabilidade, usabilidade, eficiência, portabilidade, entre outras (ROCHA; MALDONADO; WEBER, 2001).

No caminho para a obtenção de tais quesitos surgirão algumas adversidades que podem induzir o cliente a fazer uma avaliação errônea indicando baixa qualidade, como por exemplo, a ambiguidade em se definir algumas características como a manutenabilidade, que

a princípio é voltada a quem desenvolve, bem como os riscos de especificações de requisitos incompletas (SOMMERVILLE, 2003).

Conforme Pressman (2006) aponta, a qualidade de projeto e a qualidade de conformidade são os dois tipos de qualidades de software existentes. A primeira remete às propriedades projetadas para determinado item, enquanto que a segunda foca em que nível a equipe de desenvolvedores segue as especificações para o item. Essa distinção também exprime o quanto as equipes de projeto e de desenvolvimento podem impactar positivamente ou negativamente no processo de software.

É comum as empresas de software buscarem a qualidade dos produtos por meio da ampliação de funcionalidades disponíveis a clientes, eliminação de deficiências, cumprimento ou antecipação de prazos, entre outras formas. Porém é necessário ir um pouco além, introduzindo a qualidade no próprio processo produtivo. Para esse anseio algumas etapas precisam ser cumpridas, entre as quais merecem destaque:

a) **garantia e padronizações de qualidade:** a qual define padrões para o processo de software direcionados à qualidade desejada (SOMMERVILLE, 2003);
b) **planejamento de qualidade:** em que ocorre a seleção das qualidades a considerar no processo de software e a forma como ocorrerá a avaliação dos quesitos de qualidade escolhidos (SOMMERVILLE, 2003);
c) **controle de qualidade:** que visa garantir a execução dos padrões e procedimentos que regem a qualidade no processo de software (SOMMERVILLE, 2003);
d) **custo da qualidade:** na qual se quantifica o custo para a obtenção da qualidade desejada (PRESSMAN, 2006).

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

É pertinente ressaltar que as etapas destacadas na tabela 1 podem ser implementadas conforme padronizações elaboradas pela empresa desenvolvedora de software ou podem fazer parte de uma normatização técnica, que meça critérios de qualidade, conforme já exposto no capítulo anterior. A estrutura prevista nas normas ISO/IEC 9126-1, ISO/IEC 9126-2, ISO/IEC 9126-3 e ISO/IEC 9126-4, por exemplo, são aplicáveis respectivamente como modelo de qualidade, métricas externas, métricas internas e métricas da qualidade em uso (ROCHA; MALDONADO; WEBER, 2001).

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

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

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

Fonte: Adaptado de Pressman (2006).

2.1.1 Modelos de Qualidade

Implantar a qualidade é uma necessidade real e pujante nas empresas de software (SOMMERVILLE, 2003). Nada é mais frustrante do que investir pesadamente em tecnologias, estruturas operacionais, equipamentos, equipes altamente treinadas e aparato de suporte técnico ao uso do sistema, se a qualidade do produto for ínfima ou instável a cada nova alteração.

A padronização do processo de software, inserindo métricas que monitorem a própria cadeia produtiva é uma alternativa viável frente a estes desafios (ROCHA; MALDONADO; WEBER, 2001). Por isso, como já exposto no presente trabalho, entidades como a ISO/IEC (ROCHA; MALDONADO; WEBER, 2001), ANSI/IEEE (ROCHA; MALDONADO; WEBER, 2001), SOFTEX (SOFTEX, 2011) e SEI (SEI, 2010, tradução nossa) não medem esforços na produção e evolução de documentações padronizadas que regulamentam e mensuram a qualidade do processo produtivo e dos produtos de software (GUERRA; COLOMBO, 2009).

As iniciativas governamentais em parceria com organizações públicas e privadas, têm incentivado a adoção de melhores práticas no desenvolvimento dos produtos de software

por meio do PBQS, que nos últimos anos premiou vários trabalhos com objetivos alinhados ao uso de normatizações e métodos para avaliação da qualidade (BRASIL, 2006; GUERRA; COLOMBO, 2009). Entre tais normatizações e métodos destacam-se:

a) CMMI (SEI, 2010, tradução nossa);
b) MPS.br (SOFTEX, 2011);
c) ISO/IEC 9126-1 (ABNT, 2003).

Os detalhes de tais normatizações serão discutidos em subcapítulos posteriores. Cabe destacar que além de conhecer as estruturas básicas, também será dado um enfoque na etapa de verificação do produto que cada modelo comporta, com destaque para os processos de teste de software.

2.1.1.1 Os modelos CMMI e MPS.BR

Diversos modelos de maturidade, padrões, metodologias e diretrizes podem colaborar com as organizações para que estas atinjam suas necessidades de qualidade no processo produtivo e por consequência, nos produtos de software, sendo que boa parte das abordagens de melhoria tendem a atuar em uma área específica da cadeia produtiva (SEI, 2010, tradução nossa).

A proposta do modelo CMMI é de introduzir as melhores práticas em todo o ciclo de vida do software, desde a sua concepção inicial, passando pela implementação, entrega e manutenção do produto, pois contem práticas que abrangem o gerenciamento de projetos, gestão de processos, engenharia de sistemas, engenharia de hardware, engenharia de software, e outros processos de apoio utilizados no desenvolvimento e manutenção dos produtos de software (SEI, 2010, tradução nossa).

Segundo Pressman (2006), por se tratar de um metamodelo, o CMMI é adaptável às circunstâncias de trabalho de diversas empresas. Isso permite que as empresas possam optar por aplicar o modelo CMMI por completo ou ajustá-lo a características próprias de produção.

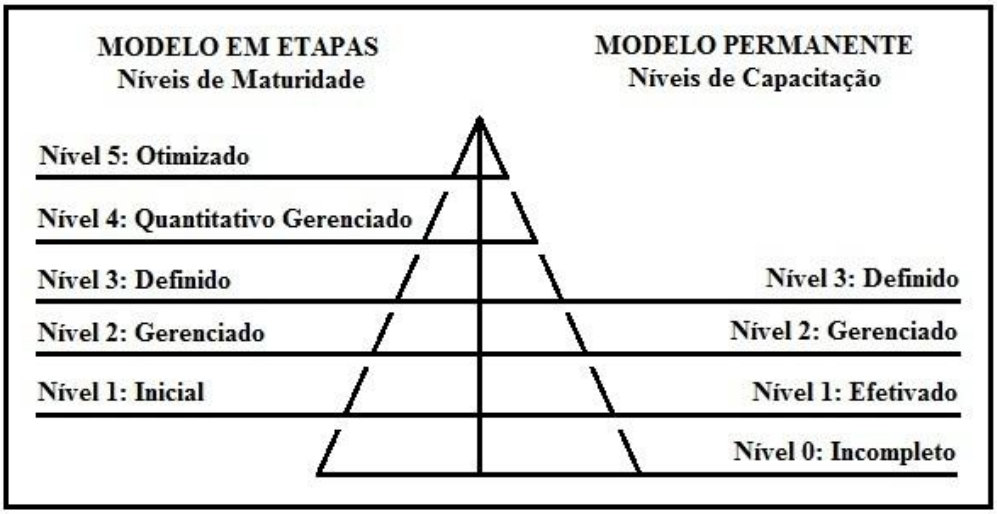
Entre os recursos que compõem o modelo CMMI estão as áreas de processos, que cumprem papel assemelhado às etapas da estrutura de processos genéricos de software (PRESSMAN, 2006) já esplanada no presente trabalho. Cada área de processo pode produzir produtos de trabalho, que são documentações, modelos, descrições de processos,

especificações de requisitos, produtos, partes de produtos, serviços, entre outros recursos, resultantes de alguma tarefa executada em determinada área de processo.

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

A figura 2 demonstra essa divisão, bem como os níveis de capacitação e maturidade pelos quais o processo de software pode ser avaliado.

Figura 2 – Modos e níveis do CMMI.



Fonte: Adaptado de SEI (2010, tradução nossa).

Para o modelo permanente, os níveis recebem as seguintes definições (SEI, 2010, tradução nossa):

a) **incompleto (nível 0):** quando determinada área do processo sequer é realizada ou se é realizada, ainda não satisfaz a níveis de capacitação exigidos pelo CMMI para o nível 1;
b) **efetivado (nível 1):** no qual os níveis de capacitação exigidos para determinada área de processo no nível 1 foram atingidos e todas as atividades estão produzindo os resultados pertinentes;
c) **gerenciado (nível 2):** em que as áreas do processo estão sob controle, as equipes envolvidas estão atuando de forma plena nas atividades, os resultados da atuação estão sendo gerados conforme o necessário, com os envolvidos tendo acesso aos recursos e ferramentas que precisam em cada tarefa, sendo que a etapa está totalmente supervisionada para garantir a plena aprovação e adesão ao escopo do processo, além de todos os critérios do nível 1 terem sido atingidos;

d) **definido (nível 3):** no qual o processo passou a ocorrer com base em diretivas da própria empresa, alimentando assim um arcabouço de processos da própria organização, colaborando com o resultado operacional pela melhoria do processo envolvido, sendo que os critérios do nível 2 foram alcançados.

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

a) **quantitativamente gerenciado (nível 4):** em que os critérios do nível 3 foram conseguidos e o processo passa a ser gerenciado de forma quantitativa, por meio de medições da qualidade e desempenho;

b) **otimizado (nível 5):** quando mecanismos estatísticos são utilizados para promover uma otimização do processo, gerando um aperfeiçoamento contínuo da área de processo que estiver sob avaliação, sendo que todas as exigências de maturidade até o nível 4 já estão atingidas.

Outra definição importante do CMMI é que as áreas de processo também estão subdivididas em categorias, as quais estão distribuídas por vários níveis de maturidade. A tabela 2 demonstra as áreas de processo, suas categorias e o respectivo nível de maturidade em que ocorrem (SEI, 2010, tradução nossa).

A tabela 2 mostra as quatro categorias existentes no processo de software do modelo CMMI. Nessa estrutura cada área do processo produtivo tem suas próprias metas, bem como práticas necessárias para o alcance de tais metas. Com isso as características de uma área do processo são definidas pelas suas metas especiais, que são refinadas pelas práticas especiais da respectiva área de processo (PRESSMAN, 2006).

Assim como em outros modelos de processo de software que visam qualidade, o CMMI usa, entre outras rotinas, processos de testes para minimizar ou eliminar possíveis erros no produto. Os testes estão inseridos na área de processo denominada de verificação, que tem como finalidade principal garantir que produtos de trabalho originados em outras áreas de processo, estejam condizentes com os requisitos funcionais desejados. Isso envolve práticas especiais como a preparação de testes, verificação de desempenho, verificação e identificação de ações corretivas, verificação do produto final e dos produtos de trabalho intermediários, além de checagens de serviços e sistemas de serviços, entre outras práticas (SEI, 2010, tradução nossa).

A verificação ocorre à medida que novos produtos de trabalho vão surgindo, por isso se trata de um processo incremental, iniciado na definição de requisitos e que evolui com o produto de software durante todo o ciclo de vida, fazendo com que a verificação amplie substancialmente a probabilidade de que o produto atenderá ao cliente (SEI, 2010, tradução nossa).

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

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

Fonte: Adaptado de SEI (2010, tradução nossa).

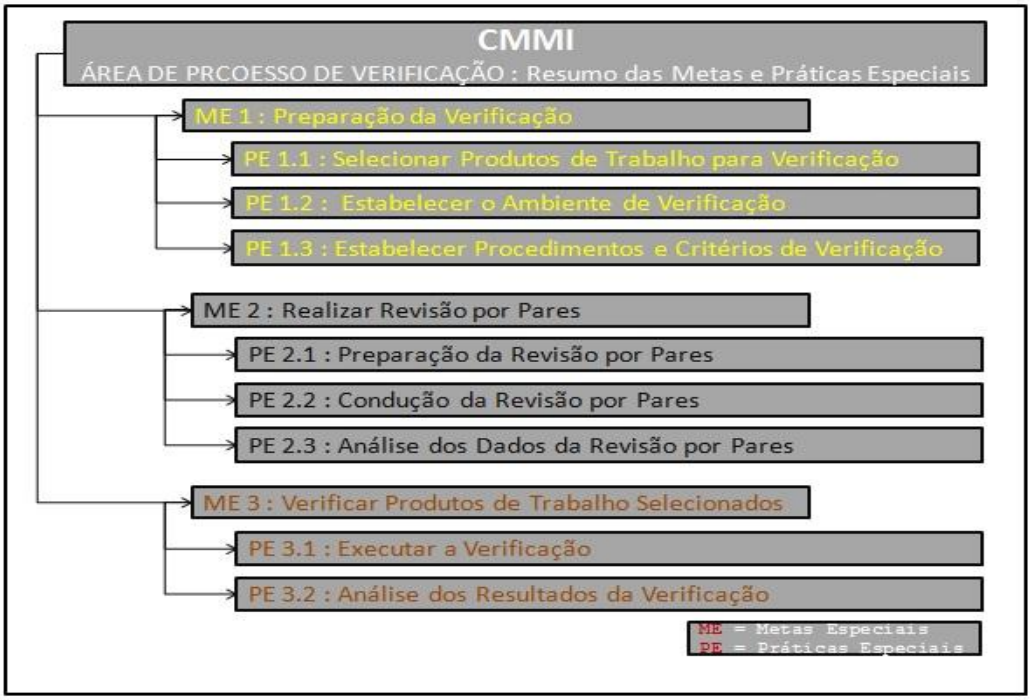
Considerando que o CMMI comporta também a área de processo de validação, convém destacar que apesar de semelhantes, verificação e validação têm objetivos distintos, o primeiro verifica se "você construiu direito" e o segundo valida se "você construiu a coisa certa" (SOMMERVILLE, 2003).

As áreas de processo do CMMI possuem uma estrutura encadeada de metas e práticas especiais. Na figura 3 é possível observar a estrutura sumarizada da área de processo de verificação, com as respectivas metas e práticas especiais definidas pelo modelo CMMI (SEI, 2010, tradução nossa).

Merece destaque uma contribuição importante que o modelo CMMI traz para a área de processo de verificação, a avaliação pelos pares, a qual permite que outros membros de equipe detectem falhas que possam estar obscuras devido a vícios ou rotinas operacionais

mal estruturadas, possibilitando uma oportunidade adicional de melhora no próprio processo produtivo do software (SEI, 2010, tradução nossa).

Figura 3 – Metas e práticas da área de processo de verificação no CMMI.



Fonte: Adaptado de SEI (2010, tradução nossa).

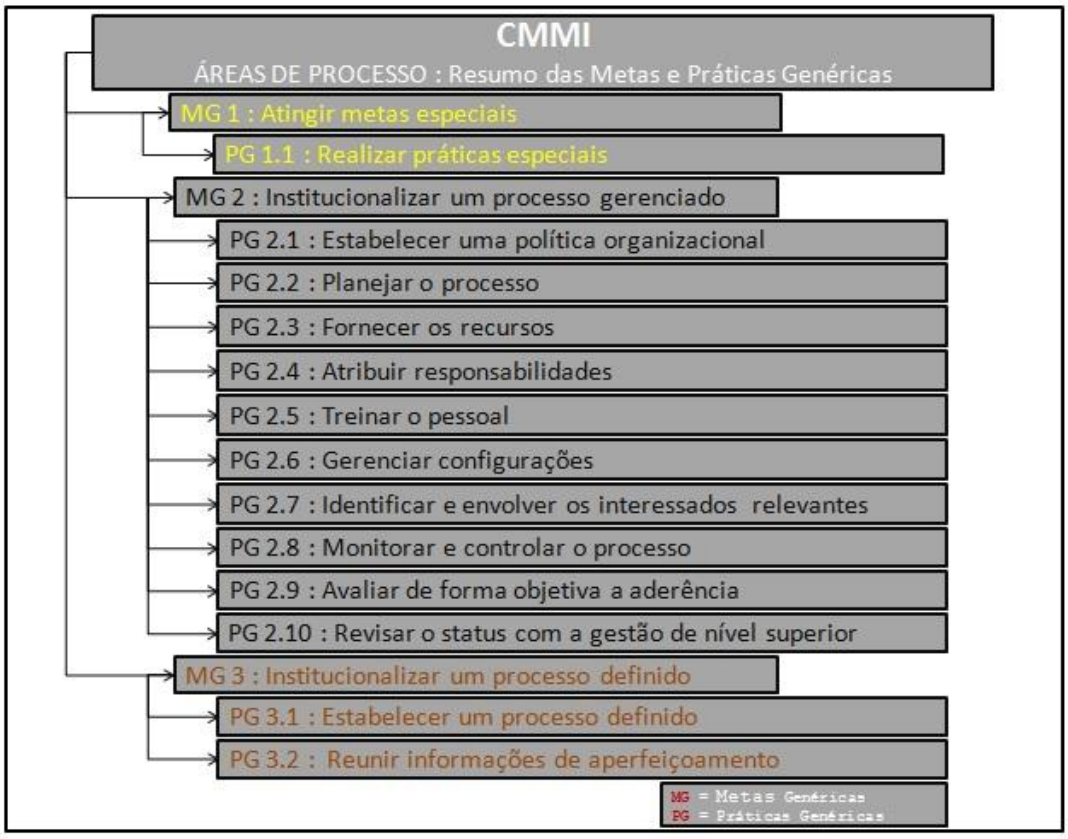
O modelo CMMI prevê também uma estrutura de metas e práticas genéricas para emprego nas áreas de processo. São três metas, cada uma com várias práticas genéricas, as quais correspondem aos níveis efetivado, gerenciado e definido de capacitação. A figura 4 apresenta de forma resumida essa estrutura genérica de metas e práticas para as áreas de processo.

Como é possível observar, o CMMI possui algumas metas e práticas especiais, aplicáveis de forma específica para algumas áreas de processo, como no exemplo da área de processo de verificação na figura 3, bem como outras metas e práticas genéricas e aplicáveis a todas as áreas de processo, conforme exposto na figura 4. As metas genéricas descrevem aspectos essenciais em que um processo pode ser institucionalizado. A institucionalização refere-se ao grau de repetibilidade, padronização e sofisticação do controle (KANUNGO; GOYAL, 2004, tradução nossa). Conclusivamente alcançar os mais altos níveis de maturidade e capacitação conduz o processo de software a uma elevação da qualidade, tanto no processo em si como no produto final.

Por ser um modelo de processo de software que demanda investimentos consideráveis tanto em reestruturação de equipes, quanto na própria reengenharia de

processos, o CMMI se mostra uma padronização que esbarra nas limitações físicas e até financeiras de micro, pequenas e médias empresas brasileiras desenvolvedoras de software (COUTO, 2007; SOFTEX, 2011).

Figura 4 – Metas e práticas genéricas no CMMI.



Fonte: Adaptado de SEI (2010, tradução nossa).

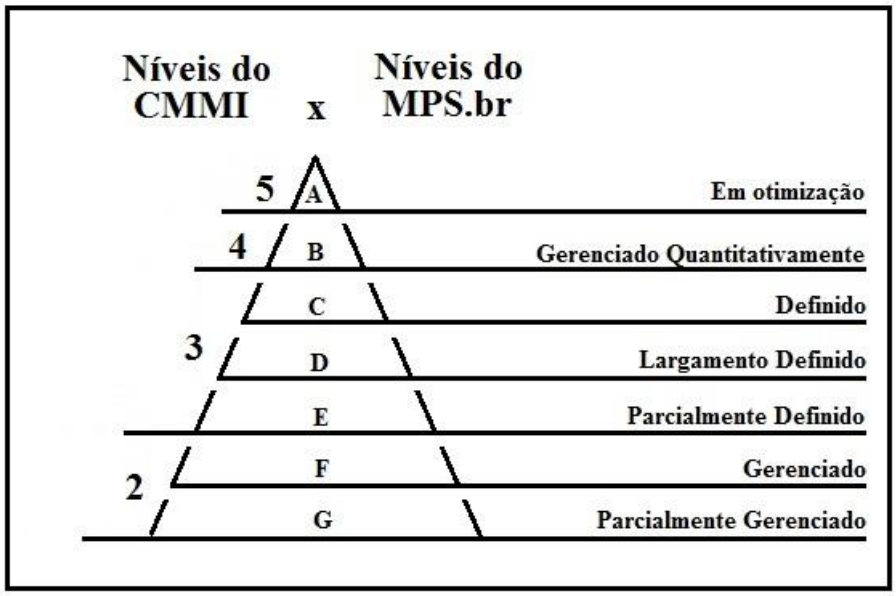
Com vistas a driblar esse cenário e possibilitar que organizações de todos os portes consigam implantar com êxito melhorias em seu processo de software, foi criado o modelo de Melhoria de Processo do Software Brasileiro (MPS.BR) (SOFTEX, 2011).

Desenvolvido e mantido pela SOFTEX, o MPS.BR é composto pelo Modelo de Referência (MR-MPS), Método de Avaliação (MA-MPS) e Modelo de Negócios (MN-MPS), recursos com os quais apoia as empresas interessadas em introduzir uma melhoria no processo de software (SOFTEX, 2011).

Conforme é possível observar na figura 5, a estrutura do modelo MPS.BR varia do nível G (mais baixo) até o nível A (mais alto) de maturidade, sendo que estes níveis são aplicáveis de forma análoga aos níveis do modelo em etapas do CMMI. A diferença está nos níveis G e E, que são respectivamente denominados de parcialmente gerenciado e

parcialmente definido. Estes dois novos níveis funcionam como sub-níveis, que facilitam o alcance dos patamares de maturidade e a transição entre os níveis, facilitando o ingresso das micro, pequenas e médias empresas na melhoria do processo de software (COUTO, 2007; SOFTEX, 2011).

Figura 5 – Níveis do modelo MPS.br X níveis do modelo CMMI.



Fonte: Adaptado de SOFTEX(2011) e SEI(2010, tradução nossa).

Considerando que o foco do trabalho atual são as rotinas ligadas ao processo de teste de software, convém citar que para o modelo MPS.BR, tal recurso é aplicado no nível D, o qual contempla a área do processo de verificação do produto (SOFTEX, 2011), também de forma análoga ao que ocorre no nível três do modelo CMMI (SEI, 2010, tradução nossa).

O subcapítulo seguinte detalhará a norma NBR ISO/IEC 9126 (ABNT, 2003), demonstrando algumas características, a contribuição de tal modelo para a qualidade de software e como o mesmo emprega o processo de testes.

2.1.1.2 O modelo NBR ISO/IEC 9126

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

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

O modelo para características internas e externas está dividido entre seis características, sendo que cada uma se subdivide em atributos que posteriormente são medidos e avaliados segundo regras da norma.

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

a) **funcionalidade:** determina se o software comporta o uso de funções com exigências explícitas ou implícitas e suas propriedades especiais;
b) **confiabilidade:** define se o software pode se manter confiável, apresentando seus resultados habituais por um determinado período de tempo e sob determinadas condições;
c) **usabilidade:** indica o nível de esforço necessário para o uso do software com base na avaliação de um grupo de usuários;
d) **eficiência:** remete à relação entre o grau de desempenho do software com o número de rotinas em uso dentro de um escopo de situações estabelecidas;
e) **manutenabilidade:** refere-se ao nível de complexidade ou facilidade em se realizar manutenções no software com baixo reflexo negativo;
f) **portabilidade:** indica o quanto um software pode ser facilmente transposto de um ambiente operacional ou uma estrutura de sistemas, para outra.

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

a) **efetividade:** determina o quanto o software possibilita aos usuários que consigam realizar tarefas específicas conforme desejam, mantendo exatidão nos resultados obtidos;
b) **produtividade:** define o relacionamento entre o número de rotinas utilizadas para se obter o máximo de efetividade;
c) **segurança:** indica se os níveis de risco oferecido pelo software a terceiros está dentro de patamares aceitáveis, frente a um contexto de uso específico.
d) **satisfação:** refere-se à forma como o software obtém avaliações de satisfação pelos usuários, também frente a um contexto de uso específico.

Conforme ocorre para outras normas, a NBR ISO/IEC 9126-1 (ABNT, 2003) deve ser considerada como uma das alternativas para a avaliação da qualidade em produtos de software, ou seja, pode ser aplicada de forma completa, quando necessário ou parcial, como recurso complementar a outros modelos de processo de software.

Mesmo com todos os recursos oferecidos pelos modelos e normatizações do processo de software, a rotina de manutenção de software continua sendo algo crítico. O próximo subcapítulo abordará esse tema, detalhando seus conceitos e principais características.

2.2 MANUTENÇÃO DE SOFTWARE

A manutenção de software é algo tão corriqueiro e necessário quanto a manutenção de qualquer outro bem, como por exemplo, de um veículo ou uma escada rolante.

Enquanto os bens materiais normalmente sofrem reparos ou trocas de peças devido ao desgaste físico decorrentes do uso contínuo, o software não recebe esse impacto por ser intangível (SOMMERVILLE, 2003). Por isso é complexo definir o quanto uma manutenção poderá acarretar de interferência involuntária nos pontos que funcionavam antes da intervenção, o que torna a manutenção de software algo diferenciado.

Segundo Pressman (2006) se gasta mais tempo dando manutenção em um software, do que para criá-lo, chegando a ser despendido cerca de 60% do esforço produtivo para essa tarefa. Naturalmente os custos chegam à marca de 60% do total sendo despendido na tarefa de manutenção (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa).

Ainda conforme Pressman (2006), a rotina de manutenção pode ocorrer por diversos fatores, entre os quais merecem destaque:

a) necessidade de atualizá-lo frente a novas tecnologias ou integrá-lo a outros sistemas;
b) alteração para incluir novas funcionalidades requisitadas pelo cliente final, impostas por órgãos governamentais ou outras entidades;
c) correção de falhas originadas nos requisitos funcionais, no projeto ou no desenvolvimento.

Considerando que as manutenções normalmente ocorrem em espaços de tempo curtos, com contratos ou órgãos legais exigindo urgência, maiores são as chances de que ocorra uma queda na qualidade do produto. Por conta disso os produtos de software que passam por manutenções constantes exigem uma maior aplicação de recursos, ferramentas, técnicas ou normas que tragam acurácia na qualidade final.

Um dos quesitos que são monitorados pelas empresas em busca da qualidade é a manutenabilidade, a qual terá diversas características expostas no subcapítulo seguinte.

2.2.1 Manutenabilidade

Garantir a qualidade em um produto de software que demanda grande manutenção é um objetivo que exige um esforço diferenciado das equipes envolvidas (SOMMERVILLE, 2003).

Segundo Rocha, Maldonado e Weber (2001), para isso a manutenção precisa ocorrer com o desenvolvedor atento aos pontos do projeto que realmente devem sofrer intervenção, reaplicando as técnicas que usou para obter a qualidade na criação original. As correções ou alterações devem estar em conformidade com os requisitos funcionais e a estrutura lógica, sendo que os demais itens que estavam em funcionamento antes da intervenção, devem permanecer operacionais. Por último deverão ocorrer testes funcionais que abranjam os pontos alterados, bem como testes de regressão das partes que não passaram por manutenção, mas que devem continuar funcionando plenamente.

Essas particularidades exigem implementações que facilitem manutenções futuras, permitindo que da forma mais transparente possível os impactos sejam rastreáveis ou previsíveis. O objetivo é garantir que qualidades, como por exemplo, a confiabilidade, se mantenha em níveis adequados (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa). A essa capacidade melhorada da manutenção do software dá-se o nome de manutenabilidade.

A manutenabilidade é composta pela facilidade dos reparos e a praticidade da evolução do software. A primeira característica é refletida principalmente pela forma modular com que ocorre o projeto e desenvolvimento do produto, por meio do reuso de componentes padrões e o emprego de rotinas da orientação a objetos. A segunda característica refere-se à minuciosidade da documentação e a adoção de técnicas, como por exemplo, a análise criteriosa do projeto original, para assim antecipar futuras manutenções evolutivas e tentar simplificá-las já na implementação que estiver em curso (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa).

A normatização NBR ISO/IEC 9126-1 (ABNT, 2003) apresenta a manutenabilidade como sendo uma característica dividida entre as seguintes subcaracterísticas:

a) **analisabilidade:** indica se um software pode ter falhas e a necessidade de melhorias rastreadas de forma fácil;

b) **modificabilidade:** aponta se o produto pode sofrer alterações específicas de maneira simples;

c) **estabilidade:** demonstra quanto é possível minimizar efeitos inesperados na manutenção do software;

d) **testabilidade:** avalia o quanto o software pode ser submetido a testes posteriores às alterações;

e) **conformidade com a manutenabilidade:** apresenta se o software está capacitado para aderir às normatizações relacionadas à manutenabilidade.

Conforme é possível observar, a norma NBR ISO/IEC 9126-1 (ABNT, 2003) também preconiza a necessidade da realização do processo de testes como recurso para assegurar qualidade, nesse caso por meio do quesito de manutenabilidade dos produtos de software. Nos capítulos e subcapítulos seguintes será abordado o processo de testes de software, com detalhamento de seus conceitos, suas técnicas, ferramentas e demais características.

3 TESTES DE SOFTWARE

O uso do processo de testes de software tem por objetivo confirmar, entre outros fatores, se as funcionalidades desenvolvidas correspondem ao que foi solicitado. Isso equivale a dizer que os testes de software buscam identificar os defeitos do produto de software (MYERS, 2004, tradução nossa; ROCHA; MALDONADO; WEBER, 2001; SOMMERVILLE, 2003).

Em um teste escolar o aluno recebe uma nota que mede o quanto ele aprendeu. As notas definem se o aluno será reprovado quanto à qualidade do conhecimento exposto no teste. De forma análoga um cliente reprovará a qualidade de um produto de software se não atender aos requisitos funcionais que espera. Nesse último caso a existência de uma equipe focada nos testes do software, antes que ele se torne operacional, atua como um filtro contra os defeitos. Devido a isso a prática do teste de software é tida como um dos principais recursos para garantia da qualidade em produtos de software (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa; MOLINARI, 2010; MYERS, 2004, tradução nossa; RIOS; MOREIRA, 2006; SOMMERVILLE, 2003).

A implantação do processo de testes exige investimentos iniciais em equipes, equipamentos e instalações que geram incertezas de que haverá algum ganho. Contudo o impacto positivo gerado pela adoção do processo de testes é descomensurável, posto que reduz o tempo de produção, diminui custos e amplia a qualidade. As consequências diretas são uma maior produtividade, ampliação da lucratividade, além da melhora na relação com o cliente final pela confiança que o produto e a equipe de produção transmitem (BASTOS et al, 2007).

A redução de tempo e custo produtivo ocorre principalmente porque o teste do software avalia desde a sua documentação, que envolve os manuais ou os próprios requisitos funcionais, até o produto do software em si. Isso refina a cadeia produtiva para que tudo esteja documentado exatamente conforme o produto será entregue, sendo que ao final do processo de software, tanto o produto quanto a documentação deverão corresponder aos requisitos funcionais desejados (RIOS; MOREIRA, 2006).

O resultado será a correção de vários defeitos ainda nas documentações ou na modelagem do sistema, o que é mais barato, pois evita impactos posteriores no tempo e custo de trabalho, restringindo ocorrências futuras para as seguintes equipes (BASTOS et al, 2007):

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

Apesar de eficaz na obtenção da qualidade, as tarefas de teste precisam ser realizadas por equipes especializadas, que conheçam estratégias e técnicas de testes (RIOS; MOREIRA, 2006), além de terem noções sobre regras de negócios que serão convertidas em requisitos funcionais do software. Essas considerações são importantes porque o nível crítico das rotinas em que o software é empregado determinará a quantidade de testes a realizar (BASTOS et al, 2007).

O teste de software emprega algumas técnicas podendo ser citado entre elas a técnica de caixa-branca, também chamada de estrutural e a técnica de caixa-preta, conhecida como funcional (PRESSMAN, 2006). Existem ainda as técnicas com base em erros e com base em máquinas de estados finitos. As técnicas são complementares entre si, cabendo um estudo das circunstâncias e das melhores formas possíveis de aplicação (ROCHA, MALDONADO; WEBER, 2001).

A realização do processo de testes está dividida em fases ou etapas, que correspondem ao teste de unidade, teste de integração e teste de sistemas. Cada uma das etapas possui finalidades específicas, como por exemplo, a etapa dos testes de unidade, que explora as menores partículas do software ou a de testes de sistemas, na qual o produto é executado na busca de confirmar a exatidão das características do software (DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006; ROCHA; MALDONADO; WEBER, 2001).

Entre as tarefas que podem ser realizadas dentro do processo de testes de software está a documentação dos roteiros e casos de testes, que norteiam a execução do processo dos testes, permitindo a avaliação das funcionalidades por meio de entradas válidas ou inválidas, para assim serem produzidos relatórios finais de teste (DELAMARO; MALDONADO; JINO, 2007).

A criação dos roteiros e casos de testes requer o emprego de critérios que definam a melhor abrangência do processo de teste, evitando que este último ocorra além ou aquém do necessário. Uma quantidade de testes acima do que seria preciso causará atrasos na entrega do

produto, por outro lado, testes em quantidade menor do que seria indispensável, produz riscos de baixa qualidade devido à quantidade dos defeitos não detectados e entregues com o produto (RIOS; MOREIRA, 2006; ROCHA, MALDONADO; WEBER, 2001). É válido observar que tais critérios de derivação dos casos de testes funcionais podem ser divididos entre formais e não formais.

Ainda conforme Rios e Moreira (2006) observam, o critério não formal baseado no conhecimento e experiência do especialista é uma opção aplicável para a derivação de casos de testes funcionais com vistas a produzir resultados satisfatórios. A informação se mostra válida considerando que a prática é comum na indústria de software da região de Criciúma.

Os critérios de derivação formais estão divididos entre particionamento de equivalências, análise do valor limite e grafo causa-efeito, estes também aplicáveis na técnica de testes funcionais, existindo ainda os critérios com base na complexidade, com base em fluxo de controle e com base no fluxo de dados para a técnica estrutural (PRESSMAN, 2006; ROCHA; MALDONADO; WEBER, 2001). Segundo Pressman (2006), pode-se acrescentar na lista o critério de teste de matriz ortogonal, a qual se aplica na técnica funcional.

A combinação destes elementos resulta em um processo de teste de software estruturado, que dinamiza a busca dos defeitos, ao passo que limita a aplicação dos testes para que ocorram dentro de um escopo aceitável de qualidade e prazo. Dando sequência aos estudos, no próximo subcapítulo será abordada a técnica de teste caixa-preta ou funcional, demonstrando suas principais características, como por exemplo, os tipos de testes suportados e limitações da técnica, além de ser apresentado em destaque o tipo de teste funcional de regressão, que também é um dos focos principais do presente trabalho.

3.1 TESTE CAIXA-PRETA

A técnica de teste caixa-preta desconsidera a estrutura interna do produto, como por exemplo, funções, procedimentos e métodos do software, baseando-se apenas nos requisitos funcionais e nas regras de negócio em que o produto se aplica (ROCHA; MALDONADO; WEBER, 2001).

A estrutura de casos de testes planejada explora, por meio de entradas válidas, as respostas válidas ou inválidas, que demonstram a aderência do software aos requisitos funcionais e ao domínio necessário dos dados. O objetivo é isolar erros ligados à omissão ou

incorreção de funções, mau comportamento de interface, falhas na estrutura ou acesso a dados, problemas de desempenho, bem como erros de iniciação e término (PRESSMAN, 2006). Ainda assim, segundo Molinari (2008), uma das limitações dos testes funcionais ou de caixa-preta é a impossibilidade de se garantir que todas as linhas do código programado ou todos os caminhos do fluxo de controle do software passarão por testes.

A técnica funcional permite o emprego de diversos tipos de testes na busca de defeitos específicos, entre eles podem ser citados os testes de (BASTOS, et al, 2007):

a) **requisitos:** buscam isolar defeitos nos requisitos do software quando comparados com o produto final obtido para a execução dos testes. Também exploram o tempo que as funcionalidades permanecerão com funcionamento adequado. Ocorrem por meio da criação de *checklists* que contenham recursos funcionais e situações de testes, sendo que as situações de testes são estruturadas à medida que o ciclo de vida do software evolui;

b) **regressão:** visam garantir que partes do software que já estavam em funcionamento antes de uma alteração ou um novo recurso criado, permaneçam em pleno funcionamento sem reflexos negativos nas funcionalidades ou no desempenho. Em subcapítulo posterior este tipo de teste será amplamente abordado diante de sua importância para softwares de manutenção constante, um dos focos do trabalho em curso;

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

d) **suporte manual:** verificam se os procedimentos de suporte manual para os testes têm documentações e se estão completos. Também valida se as responsabilidades pelo suporte manual estão claramente definidas na equipe. Devem ocorrer já nas fases iniciais do ciclo de vida do software, porém testes extensivos devem ser realizados em paralelo com as etapas de entrega e implantação do software;

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

f) **controle:** têm a finalidade de explorar se os dados estão íntegros, se as transações são realizadas adequadamente e se dados têm sido recebidos para uma possível auditoria de processos. Também avalia se os processos são realizados de forma eficiente, econômica e atendendo às necessidades funcionais. São aplicados com o desenho de controles que minimizam riscos no software, sendo considerado quase que um sistema dentro de outro. Após isso o responsável pelos testes cria situações de risco para conferir o resultado dos testes de controle. Um dos métodos utilizáveis é a matriz de riscos, que contém também os controles e o segmento do sistema em que se aplicam;

g) **versões paralelas:** visam a comparação entre uma nova versão de software com sua versão anterior, validando ou invalidando as funcionalidades previstas para a nova versão. Podem exigir a atualização dos dados de entrada conforme os padrões esperados pela nova versão em teste. Sucedem com a execução simultânea da versão nova e a versão anterior, com análise comparativa de resultados obtidos nas funcionalidades testadas, tentando isolar possíveis defeitos.

Entre todos os tipos apontados, o teste de regressão se mostra uma opção viável e aplicável para softwares de constante manutenção, desempenhando papel de destaque na redução de prazos e custos, bem como na ampliação da qualidade de tais produtos de software, sendo praticado manualmente ou com aplicação de recursos da automação de testes. Os ganhos ocorrem por meio da reexecução de casos de testes com resultados válidos para pontos do software que não sofreram modificações, mas que precisam prosseguir operando perfeitamente, como operavam antes da intervenção (PRESSMAN, 2006).

O conhecimento sobre os tipos de testes permite avaliar o que será mais adequado ao contexto de cada software que se pretenda submeter a um teste funcional ou caixa-preta, bem como frente ao que se pretenda investir de tempo e recursos para a obtenção de ganho na qualidade da produção. Procurando manter essa visão, o subcapítulo seguinte abordará as definições dos principais critérios formais e não formais para a derivação de casos de teste caixa-preta.

3.1.1 Critérios Para Derivação de Teste Caixa-preta

A elaboração de roteiros e casos de testes caixa-preta por meio de abordagens sem uso de critérios pode produzir testes que ampliam o tempo do processo de software ou culminar com sequências de testes aquém do necessário, escondendo defeitos e impactando na qualidade do produto.

Também é válido considerar que, segundo Myers (2004, tradução nossa), testes exaustivos são impossíveis ou impraticáveis, pois o número de variações de testes que cubra todas as combinações de entradas válidas ou inválidas tende a onerar o processo.

Diante destas limitações e necessidades a engenharia de software provê o uso de critérios para o levantamento dos roteiros e casos de testes.

Os roteiros de testes estão ligados mais aos requisitos funcionais que devem ser submetidos a testes no software. Os casos de testes são direcionados à forma como ocorrerá o teste de tais requisitos, com as respectivas entradas válidas ou inválidas, bem como as saídas esperadas (MOLINARI, 2008).

Entre os critérios mais relevantes para seleção dos roteiros e casos de testes estão os de:

a) **particionamento de equivalência:** as entradas válidas ou inválidas que serão submetidas a testes são agrupadas em classes de equivalências. Cada classe passa a representar um grupo do domínio de entrada, sendo que se um dos elementos do grupo possuir um determinado comportamento válido ou inválido, os demais são considerados equivalentes a ele. Em suma o número de variações em roteiros ou casos de testes tende a diminuir posto que o teste de um elemento de cada grupo corresponderá a submeter todo o grupo ao mesmo teste (DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006);

b) **análise do valor limite:** é um critério complementar ao particionamento de equivalência, porém trabalha com áreas limítrofes de domínio para as entradas e saídas. A aplicação do critério de análise do valor limite ocorre sobre as partições de equivalência, produzindo roteiros ou casos de testes que explorarão os limites de domínio por classes (DELAMARO; MALDONADO; JINO, 2007). De maneira resumida, este critério também reduz a quantidade de roteiros e casos de testes ao restringir os testes às fronteiras das classes equivalentes (MOLINARI, 2008);

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

d) **matriz ortogonal:** é um critério que preferencialmente deve ser aplicado diante de domínios de entrada que inviabilizam testes exaustivos. A principal vantagem deste critério é permitir a detecção de defeitos oriundos de lógica defeituosa no software (PRESSMAN, 2006). A decisão pelo uso desse critério deve considerar que é custoso testar tudo, mas incorreto não testar nada. Para uso do critério é preciso inicialmente determinar quantas e quais condições ou combinações serão testadas. Posteriormente define-se o número máximo de valores que cada elemento condicionado ou combinado receberá. Seguindo adiante é preciso determinar quantas e quais serão as variáveis a serem tratadas no teste. Com todos os dados coletados, monta-se uma tabela na qual o valor possível para cada variável deva estar presente pelo menos uma vez. A

partir da tabela os roteiros e casos de testes serão elaborados, garantido que cada um dos parâmetros de entrada tenha seus valores possíveis submetidos a testes funcionais pelo menos uma vez (MOLINARI, 2008);

e) **experiência e conhecimento do especialista de teste:** além dos critérios formais já apresentados, os quais também têm relevância e aplicabilidade, existe também o critério não formal baseado no conhecimento e experiência do especialista de testes (RIOS; MOREIRA, 2006). O critério é empregado pelo especialista, que se vale dos conhecimentos em técnicas de testes, regras de negócios e funcionalidades a testar, além da experiência de testes anteriores e erros já detectados. Com o uso desses recursos ocorre o levantamento dos casos de testes que tenham coesão com as funcionalidades, que se mostram mais propensos a identificar erros, que sejam reutilizáveis e que não repitam testes entre si. Por fim a análise de tais fatores deve resultar em um planejamento de testes, documentado em uma matriz ou em um software de automação do gerenciamento do teste. A documentação gerada deve representar um conjunto de casos de testes coesos, eficientes e econômicos frente às funcionalidades que serão exercitadas.

Uma característica importante e inerente a todos os critérios apontados e que merece destaque, é a extrema necessidade de que os requisitos funcionais estejam completos, íntegros e coesos no momento da estruturação dos roteiros e casos de testes, pois do contrário, os resultados dos testes serão questionáveis (DELAMARO; MALDONADO; JINO, 2007)

É possível obsevar que cada um dos critérios apontados possui possibilidades distintas de aplicação, permitindo que vários deles sejam combinados para uma melhor produção de roteiros e casos de testes. Contudo para manter o foco didático, o trabalho atual empregará o uso do critério não formal baseado no conhecimento e experiência do especialista, uma vez que este é o mais difundido comercialmente na região de Criciúma, visando assim a obtenção de resultados mais realistas, que possam servir de base sólida e comparativa em futuras pesquisas envolvendo os demais critérios.

3.2 TESTE CAIXA-BRANCA

O teste caixa-branca, também chamado de teste estrutural, consiste na técnica em que os casos de testes são criados considerando apenas os recursos lógicos internos do

software (MYERS, 2004, tradução nossa), como por exemplo, estruturas de dados e fluxos de controle (PRESSMAN, 2006). O objetivo principal é assegurar que o software possua robustez e coesão na estrutura lógica de procedimentos, funções, métodos e demais recursos programados (BASTOS et al, 2007).

O software possui diversas estruturas de decisão, portanto diversas vias nas quais os dados e comandos fluirão a caminho de resultados válidos ou inválidos, ou seja, a caminho do funcionamento perfeito ou dos defeitos, sendo impraticável o teste de todas estas vias possíveis (MYERS, 2004, tradução nossa). Por isso a técnica de testes estruturais também prevê o emprego de critérios para a seleção dos casos de testes.

A técnica caixa-branca comporta vários tipos de testes específicos. Entre eles podem ser citados os testes de:

a) **estresse:** nos quais o software é submetido a execuções que testem os extremos de processamento, memória ou capacidade para leitura de dados;
b) **execução:** em que são checadas questões de desempenho e performance frente a padrões pré-estabelecidos no projeto;
c) **recuperação:** cujo objetivo é avaliar a capacidade de reversão de uma condição catastrófica ou caótica do software, como por exemplo, o retorno de um *backup* de banco de dados que tenha sofrido um dano físico ou estrutural;
d) **operação:** que visam avaliar se os operadores, com base na documentação de operação, conseguem por o software em funcionamento no ambiente de produção de maneira satisfatória e completa;
e) **conformidade:** os quais avaliam se as normas, padrões e manuais de desenvolvimento foram adotados durante a produção do software;
f) **segurança:** nos quais são avaliados os recursos que garantem a integridade e confidencialidade dos dados e informações;

É importante também destacar que os testes estruturais podem ser afetados por algumas limitações, como por exemplo, os problemas de (DELAMARO; MALDONADO; JINO, 2007):

a) **resultados para caminhos distintos:** não é possível provar que dois caminhos distintos para fluxo de dados ou de controle podem computar a mesma função;
b) **garantia da execução:** é indefinível que existam entradas de dados que asseguram a execução de um determinado caminho no fluxo de controle ou de dados do software;

c) **caminhos ausentes:** é impossível garantir que caminhos ausentes, por falha no projeto ou no desenvolvimento, sejam detectados nos testes;

d) **correção coincidente:** não há garantias de que uma entrada de dado que gere resultado válido produzirá o mesmo resultado se outro dado for imputado como entrada. Tal problema também pode estar presente em outras técnicas e em diversos tipos de testes.

Ainda segundo Delamaro, Maldonado e Jino (2007), mesmo diante destas limitações, estudos têm demonstrado que é viável o uso da técnica de testes estruturais na busca da ampliação da qualidade nos produtos de software, cabendo aos responsáveis pelas equipes de testes um estudo formal e minucioso em busca da sua melhor aplicação em complemento ao uso de outras técnicas, como a funcional, já exposta em capítulo anterior.

O próximo subcapítulo abordará questões ligadas a modelos do ciclo de vida do teste de software, suas variações, as respectivas aplicações e ganhos proporcionados, além de outros aspectos.

3.3 CICLO DE VIDA DO TESTE DE SOFTWARE

O processo de teste de software pode ser aplicado por meio de um modelo representado por uma sequência estruturada de técnicas, etapas e rotinas, a qual é denominada de ciclo de vida do teste de software. Este modelo busca dinamizar o processo e assegurar que cada passo seja completado de forma íntegra, entregando à rotina seguinte tudo o que for necessário para a continuidade dos testes com o máximo de qualidade (RIOS; MOREIRA, 2006).

Convém ressaltar que a aplicação de um modelo resulta na materialização de algum conceito que se pretenda empregar (MOLINARI, 2008).

Segundo a SOFTEX (2011), a tendência normal é de que o modelo de ciclo de vida definido para os testes de software seja aplicado de forma alinhada ao modelo adotado para o ciclo de vida do processo de software, buscando-se assim uma concomitância entre os processos. Em outras palavras, em um ciclo de vida de software incremental, por exemplo, os testes teriam seus requisitos definidos no mesmo momento em que os requisitos funcionais fossem definidos para o produto e assim sucessivamente, até o fim do processo de teste do software, de modo que tanto o produto quanto o processo tenham a qualidade verificada constantemente.

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

a) **em cascata ou *waterfall:*** modelo análogo ao seu correspondente para o ciclo de vida do software, ocorrendo de forma retilínea e sequenciada, com a verificação e validação sendo aplicadas em estágios distintos, impedindo por exemplo, que dados técnicos do produto sejam antecipados;

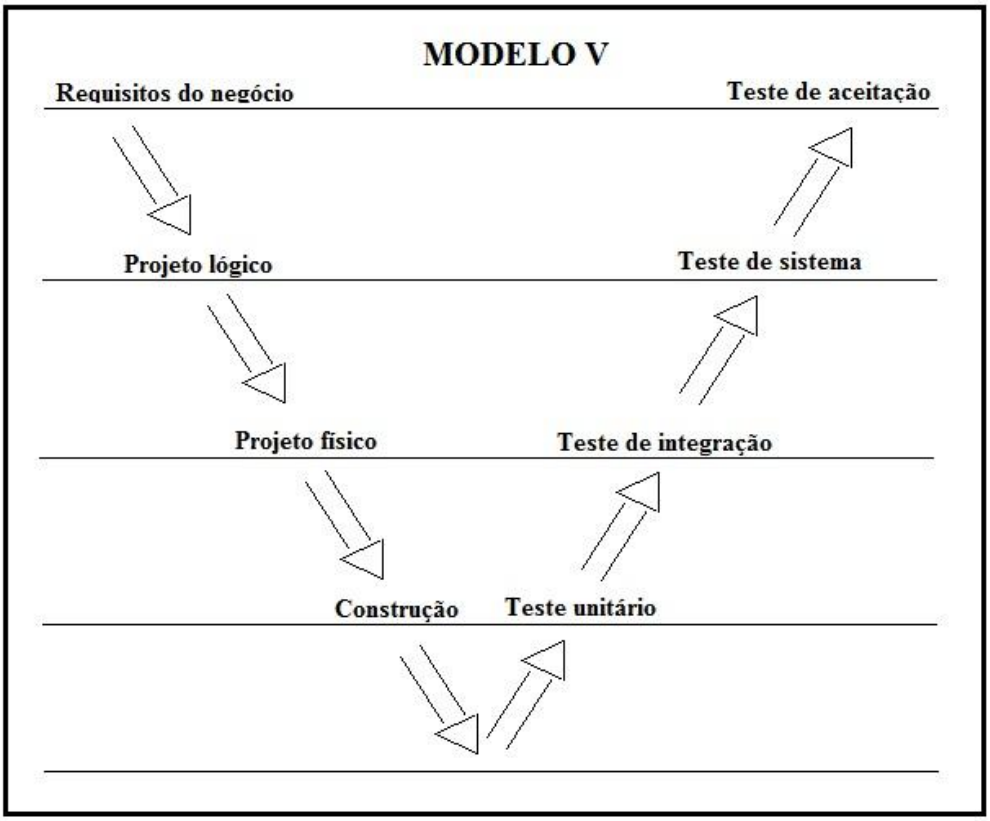
b) **modelo V:** é considerado um dos mais simples e aplicáveis em projetos variados, contudo possui como falha a fraca acoplagem de etapas permitindo por exemplo que erros de comunicação resultem em erros na definição de requisitos funcionais, o que conforme já exposto em capítulos e subcapítulos anteriores, será desastroso para o custo e o prazo do projeto se tais falhas só forem percebidas após o ciclo de vida do software ter caminhado para etapas seguintes;

c) **modelo espiral:** comporta os atos de desenvolver e submeter a testes a parte implementada do produto de software. Surgiu com o advento da *Unified Modeling Language* (UML) que propicia iteratividade no desenvolvimento de aplicações.

A figura 6 demonstra de forma esquemática o modelo V, na qual se pode observar a fluência das rotinas do processo de teste de software desde a aquisição dos requisitos funcionais até os testes de aceitação pelo cliente final (RIOS; MOREIRA, 2006).

Tendo o intuito de apresentar o ciclo de vida do processo de testes de software de uma forma mais didática, Rios e Moreira (2006) propuseram o modelo 3P x 3E, o qual emprega algumas rotinas do processo de teste continuamente e de forma paralela, enquanto que outras tarefas fluem sequencialmente com formato assemelhado ao modelo em cascata. A figura 7 demonstra essa estrutura, sendo possível a visão de como tais tarefas se integram.

Figura 6 - Modelo V para ao ciclo de vida do teste de software.



Fonte: Rios e Moreira (2006).

Ainda segundo Rios e Moreira (2006), na figura 7 é possível observar que o modelo 3P x 3E permite compreender como as tarefas de planejamento e de preparação dos testes ocorrem continuamente, enquanto que os procedimentos iniciais, especificação, execução e entrega acontecem de maneira sequencial entre si, porém paralelas à preparação e ao planejamento. Cada um dos elementos acima possuem as seguintes finalidades específicas dentro do modelo 3P x 3E do processo de testes sugerido (RIOS; MOREIRA, 2006):

a) **procedimentos iniciais:** comporta a definição de um acordo de nível de serviço, bem como um escopo da estratégia de testes que será empregada;

b) **planejamento:** será aplicado como recurso de apoio ao processo, portanto tem a finalidade de garantir o perfeito planejamento das tarefas de especificação, execução e entrega;

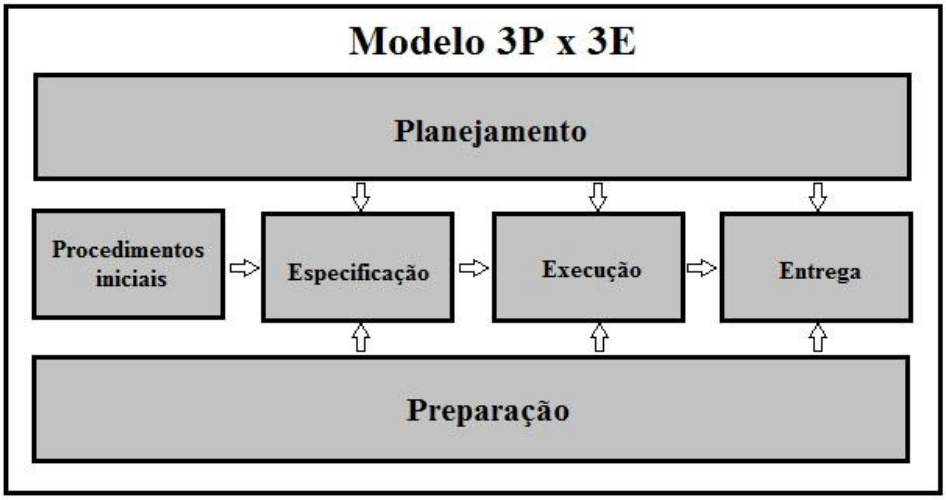
c) **preparação:** tem a finalidade de apoiar o processo assegurando que documentações, ambientes, ferramentas, softwares e o pessoal envolvido estejam plenamente preparados e disponíveis quando uma etapa de testes for posta em execução;

d) **especificação:** é a etapa na qual serão elaborados e revisados os roteiros e casos de testes;

e) **execução:** os roteiros e casos de testes são postos em execução, empregando os recursos já entregues pela tarefa de preparação. O uso de ferramentas de automação poderá ser um dos recursos aplicáveis;

f) **entrega:** comporta a finalização dos testes, com documentação dos resultados obtidos que possam melhorar o processo produtivo, arquivamento da documentação do produto e disponibilização do software para os usuários finais. Um relatório gerencial das melhorias com base em conformidades ou não conformidades também poderá ser elaborado.

Figura 7 – Modelo 3P x 3E para o ciclo de vida do processo de testes de software.



Fonte: Rios e Moreira (2006).

Apesar de existirem ainda outros modelos prontos de ciclo de vida dos testes de software, um dos recursos que podem ser empregados é a criação de um modelo adaptado às características da empresa desenvolvedora ou do produto a ser testado. Esse conceito parte da premissa de que modelos prontos podem ter suas capacidades subutilizadas diante da complexidade de um produto ou superutilizadas frente à simplicidade de outro software. Os modelos prontos também podem ser afetados pelos recursos estruturais que a empresa desenvolvedora disporá para o processo de testes, que podem ser ínfimos ou amplos tanto em equipes quanto em equipamentos e softwares de apoio. Para a criação de um modelo próprio é necessário o domínio de boas práticas em testes de software, o conhecimento daquilo que se pretende priorizar e atingir com os testes e por fim é imprescindível a identificação do melhor momento para implantar o modelo criado (MOLINARI, 2008).

A criação de um modelo próprio não é uma tarefa simples, porém é possível e como visto, pode ser viável diante de algumas circunstâncias. Para isso Molinari (2008) sugere o uso de um metamodelo, o qual pode simplificar a criação de modelos próprios. Essa simplificação ocorre considerando que tanto técnicas, metodologias e etapas são todos tipos distintos de testes. Por fim todos estes tipos de testes são distribuídos em dimensões dentro do metamodelo. Tais dimensões são estruturadas em:

a) **meta:** para o metamodelo corresponde a tipos de testes que têm objetivos diretos como teses funcionais, de performance ou de usuário. É considerado o grupo de testes que dita o "o quê" na qualidade do software;

b) **momento:** comporta os tipos de testes que têm um momento exato e correto para sua ocorrência, como os testes unitário, de integração ou de sistemas. Constitui-se do grupo de testes que determina "o quando" para a qualidade do produto;

c) **técnica:** representa o grupo de tipos de testes que distinguem técnicas distintas de abordagens do teste e o seu foco quanto ao produto, como por exemplo os testes caixa-preta ou caixa-branca. Abrange o grupo de testes que define "o como" para a obtenção da qualidade no software testado;

d) **ambiente:** comporta conceitos e meios físicos que determinam o local da prática dos testes, como por exemplo, ambiente de testes para internet ou uso de servidores mainframes. Agrupa os testes que especificam "o onde" na busca da qualidade produtiva.

A aplicação do metamodelo ocorrerá com o levantamento das necessidades de testes, posteriormente estas são distribuídas dentro do metamodelo e por fim são realizadas derivações que criam o modelo necessário em uma sequência estruturada, a qual deve garantir que os tipos de testes estejam acoplados aos momentos adequados do projeto em que devem ser executados (MOLINARI, 2008).

Indiferente ao modelo a ser adotado para o ciclo de vida do teste de software é importante que os produtos sejam estudados, as necessidades de testes elencadas, sendo que deverá ser implantado o modelo que representar menor risco, maior desempenho produtivo e aumento na qualidade. É necessário também considerar que tão importante quanto esse caminho, é o uso de avaliações permanentes da cadeia produtiva, tentando evoluí-la sempre que possível frente a novas necessidades ou possíveis falhas que forem detectadas.

Os subcapítulos posteriores detalharão as fases de testes, as quais são elementos importantes que também compõem o ciclo de vida do teste de software.

3.3.1 Fase de Unidade

O desenvolvedor aplica testes no menor elemento do software, checando os resultados, limites funcionais e a lógica de cada componente empregado no produto, como por exemplo, de funções, procedimentos ou métodos (DELAMARO; MALDONADO; JINO, 2007).

A aplicação dos testes ocorre por meio do desenvolvimento de um falso-controlador, também chamado de *driver*, que simula o controle de um falso-controlado, denominado de *stub*. Os dois elementos têm a finalidade de simular testes que exercitem os limites ou comportamentos do ponto do software a que os componentes se aplicam. Para isso casos de testes são imputados pelo falso-controlador ao falso-controlado e os resultados importantes são documentados. Como estão inseridos dentro do contexto de desenvolvimento, os elementos do teste de unidade são considerados parte do software, representando também despesas indiretas do desenvolvimento do produto (PRESSMAN, 2006).

3.3.2 Fase de Integração

Compreende a fase na qual o desenvolvedor testa a acoplagem de todos os componentes do software, entre eles o banco de dados, outros sistemas, bibliotecas auxiliares ou módulos (PRESSMAN, 2006).

A iteratividade entre os componentes do software precisa ser avaliada para garantir que juntos atuam de forma normal, sem apresentar defeitos. É um teste que requer amplo conhecimento estrutural do software, daí a necessidade de ser realizado preferencialmente pela própria equipe de desenvolvimento (DELAMARO; MALDONADO; JINO, 2007).

A fase de integração pode ocorrer por diversos meios, tendo destaque as seguintes abordagens (PRESSMAN, 2006):

a) ***big-bang*:** na qual o software tem seus componentes acoplados com base em uma combinação anterior e todos eles são testados de uma só vez, resultando em um grupo de erros que fatalmente conduzirá a um resultado caótico,

gerando a necessidade de várias sessões de testes de integração até que o produto se torne usual;

b) **incremental:** em que o produto de software tem sua acoplagem testada gradualmente, à medida que os componentes vão sendo desenvolvidos, produzindo uma lista menor de erros, acelerando as etapas de correção e as novas etapas de testes de integração. Os testes de integração estão subdivididos entre as estratégias descendente ou ascendente, sendo que a primeira ocorre com a integração de forma descendente na hierarquia de dependência dos módulos ou componentes, enquanto que a segunda ocorre de maneira ascendente partindo dos componentes mais baixos na linha de hierarquia até os mais altos.

Indiferente da estratégia adotada no teste de integração, o resultado obtido tende a ser um produto integrado, usual e pronto para ser submetido à etapa posterior de testes (PRESSMAN, 2006).

3.3.3 Fase de Sistema

A fase dos testes de sistema contempla o produto já com seus componentes integrados e estes são exercitados com a verificação de que cada um cumpre suas funções de maneira completa e satisfatória (PRESSMAN, 2006).

Essa vistoria da integralidade das funcionalidades é realizada com a execução dinâmica do software, checando por meio de entradas válidas se os resultados obtidos nas saídas condizem com o esperado para cada entrada válida e se permanecem funcionando perfeitamente diante de entradas inválidas (ROCHA; MALDONADO; WEBER, 2001).

Segundo Pressman (2006), diversos testes podem ser empregados para a realização dos testes de sistemas, podendo ser citado entre eles o teste de recuperação, de segurança, de estresse e de desempenho. Também é uma característica dos testes de sistemas a possibilidade de que ocorram dentro de mais de uma técnica e em outras fases, a exemplo do teste de desempenho, que pode ser executado dentro da etapa de teste unitário.

3.3.4 Fase de Regressão

É a fase na qual o software já está em uso pelo cliente final e após uma manutenção, passará por testes assegurando que tanto as funcionalidades implantadas pela nova intervenção no produto, quanto os demais recursos que estavam em pleno funcionamento, permaneçam plenamente operacionais ao fim do processo de testes (DELAMARO; MALDONADO; JINO, 2007; PEZZÈ; YOUNG, 2008).

Conforme Pressman (2006), o teste de regressão pode ser considerado também um dos tipos de testes aplicáveis à fase de testes de integração, por permitir que componentes ou funções sejam exercitadas individualmente para assegurar que seu funcionamento esteja coerente com o projeto.

Também é possível considerar que a regressão é um tipo de teste aplicável à técnica funcional ou caixa-preta (BASTOS et al, 2007), na qual o software será submetido a testes que exercitem as funcionalidades projetadas para o produto como um todo, tanto para as modificações, como para os pontos definidos no projeto original.

Indiferente à visão aplicada, os testes de regressão são fundamentais para que a qualidade do software seja mantida (BASTOS et al, 2007; DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006), dado que as manutenções frequentes invariavelmente afetam o software pela quantidade de defeitos que imputam ao produto (SOMMERVILLE, 2003). Este é um fator que torna a fase de regressão fundamental para o processo de testes em produtos que passam por frequentes adaptações, como por exemplo, softwares com requisitos funcionais definidos por legislações fiscais, tributárias ou trabalhistas, áreas estas que constantemente sofrem alterações, tornando importante o emprego dos testes de regressão na busca da melhoria na qualidade dos produtos.

A aplicação do teste caixa-preta ou funcional na fase de regressão, conforme já exposto, produz ganhos consideráveis em termos de qualidade do software, contudo, conforme também já foi observado em capítulos anteriores, os testes manuais possuem limites que tendem a onerar a capacidade operacional das equipes de testes frente ao tamanho dos softwares, a quantidade de requisitos funcionais a validar e as frequentes manutenções realizadas. Com vistas a isso o capítulo seguinte abordará questões ligadas à automação de testes funcionais, apresentando o conceito, ferramentas, sua aplicação, além de outras características.

4 AUTOMAÇÃO DE TESTES FUNCIONAIS

O teste de software é um processo realizado manualmente, visando identificar defeitos no produto, enquanto que a automação de testes compreende a execução das mesmas rotinas, porém de forma automatizada, com um computador e um software que simulam as entradas válidas ou inválidas que um usuário faria de forma manual, para assim avaliar o produto de software (FEWSTER; GRAHAM, 1999, tradução nossa).

O emprego da automação de testes ganha importância à medida que cresce a busca pela qualidade nos produtos de software ou pela necessidade de se reduzir prazos e custos produtivos. Contudo o uso da automação deve ocorrer somente em empresas de software que possuam uma estrutura madura, contendo uma experiente equipe de testes manuais, conhecedora do produto e de técnicas ou ferramentas adequadas de testes (MOLINARI, 2010; RIOS; MOREIRA, 2006). Aqui é importante considerar que uma vez gerados os casos de testes automáticos, alguém terá que validar os resultados iniciais como corretos, bem como será preciso avaliar possíveis erros do software ou falhas de *scripts*, para assim definir o que ajustar e como ajustar. Portanto, sem o apoio de uma equipe madura de testes manuais o trabalho da equipe de automação ficará exposto a riscos maiores.

Por outro lado, conforme Molinari (2010) expõe, a equipe de especialistas em automação de testes deve ser composta por membros que conheçam de programação e que tenham facilidade em trabalhar com lógica. Tais conhecimentos serão importantes para desenvolver ferramentas auxiliares, criar e manter casos de testes automatizados, pesquisar novas técnicas ou tecnologias de apoio, assim como para compreender toda a estrutura de automação envolvida.

É importante observar que a automação não pode ser considerada o remédio para todos os males no processo de testes de software (MOLINARI, 2010), visto que seu uso deverá ocorrer partindo de um minucioso estudo sobre o que se pretende automatizar, destacando casos de testes reutilizáveis e que envolvam pontos críticos do software que carecem de testes permanentes (FEWSTER; GRAHAM, 1999, tradução nossa). O uso de testes automatizados também pode ser empregado em situações que seriam onerosas se realizadas manualmente, como por exemplo, testes de performance que exijam muitos usuários conectados simultaneamente (MOLINARI, 2008).

Resumidamente pode-se observar que os casos de testes mais recomendados para a automação possuem as seguintes características (MOLINARI, 2010):

a) **reutilização:** compreende os casos de testes que podem ser reutilizados, considerando que em um teste anterior o caso obteve resultado válido. Isso dá garantias de que um recurso do software testado por meio do caso está com funcionamento pleno em testes posteriores ao primeiro;

b) **otimização do tempo de teste:** abrange os casos de testes que mesmo sendo pontual e específico de uma versão, exigirão menor tempo para automatizar, se comparado ao que seria necessário para preparar e realizar os testes de forma manual. Para estes casos é importante a adoção de estudos e estimativas que validem o ganho de tempo que a automação tende a conceder;

c) **otimização no uso de recursos:** envolve os casos de testes que se executados manualmente consumirão muitos recursos físicos ou de pessoal. O exemplo já citado anteriormente de testes de performance, envolvendo muitos usuários, se aplica a esta característica de casos de testes a serem automatizados. Casos de testes assim exigem uma atenção especial quanto ao montante de trabalho que a automação exigirá. O objetivo é estimar a viabilidade da ação.

Segundo Fewster e Graham (1999, tradução nossa), é comum as empresas considerarem que automatizar testes será de imediato mais rápido e barato do que testar manualmente, contudo trata-se de um ledo engano, posto que somente na segunda utilização de um teste automatizado é que haverá um ganho real. Isso ocorre porque antes do primeiro uso será preciso criar e gravar a automação dos casos de testes.

É necessário considerar também que de início os custos serão altos, tanto para montar a equipe de automação, o que envolve contratações e treinamentos, quanto para aquisição dos demais recursos, como por exemplo, equipamentos, softwares e outras tecnologias que forem necessárias para o funcionamento da equipe de automação (MOLINARI, 2008).

A automação de testes pode ser praticada em diversos âmbitos, desde o gerencial até a análise de defeitos já ocorridos. Cada automação pode empregar softwares específicos, entre os quais merecem destaque as ferramentas de (MOLINARI, 2010):

a) **plano de testes:** compreende softwares com a finalidade de facilitar a projeção e criação de roteiros e casos de testes, alinhados aos requisitos funcionais ou necessidades específicas de teste;

b) **automação dos casos de testes:** comporta softwares com os quais a equipe de automação grava a execução dos casos de testes para posterior reexecução. Os softwares com essa finalidade estão divididos em duas categorias,

- *Graphical User Interface* (GUI) *Test Drivers* com *Command Script*: possibilitam a criação de *scripts* com inserção de comandos decisórios no caso de teste que for gravado,
- *Graphical User Interface* (GUI) *Test Drivers* com *Visual Script*: comportam a criação dos *scripts* de maneira visual, sem intervenção programada. Há softwares que aceitam alguns comandos no *script* gerado;

c) **automação de testes de carga e performance:** empregada para simular uma utilização pesada do software, que exija muitos usuários ou para a alimentação e manipulação de ampla massa de dados;

d) **gerência da automação:** ferramenta que dá suporte ao gerenciamento das automações de casos de testes desenvolvidas;

e) **cobertura de código:** utilizada para avaliação do código programado pela equipe de automação de testes, tentando assim refinar a qualidade da automação desenvolvida;

f) **análise da base de conhecimento:** proporciona estudar defeitos já detectados, permitindo estudos para criar novos casos de testes ou melhorar os existentes;

g) **testes unitários:** abrange softwares empregados na automação da fase de testes unitários, podendo existir à parte ou incorporado dentro do produto de software, por meio de um arquivo *Dinamic-link Library* (DLL) ou de um método programado.

A adoção das ferramentas citadas depende do foco que se pretende aplicar ao projeto de automação de testes, assim para o trabalho atual, que visa demonstrar a cobertura funcional com testes automatizados em softwares com manutenções frequentes, os estudos focarão as ferramentas de automação dos casos de testes.

Como já exposto em capítulos e subcapítulos anteriores, a técnica de teste funcional possui uma abrangência considerável sobre os requisitos do software (SOMMERVILLE, 2003), ao passo que realizar testes exaustivos é algo impraticável para a maioria dos produtos de software (MYERS, 2004, tradução nossa). Diante disso, a automação dos testes funcionais ganha importância, pois a médio ou longo prazo reduz

incomensuravelmente o tempo dos testes, além de ampliar a qualidade do produto ao alinhá-lo a seus requisitos funcionais (PEZZÈ; YOUNG, 2008).

Os testes funcionais automatizados se valem da experiência e conhecimento dos testadores manuais (MOLINARI, 2008), porém a criação dos testes automáticos tende a ocorrer pela parceria entre as duas equipes (MOLINARI, 2010).

Uma vez documentados os casos de testes funcionais realizados manualmente e que representam resultados válidos, ou seja, pontos do software com funcionamento adequado, a equipe de automação passa a atuar na gravação desses casos e geração do respectivo *script*, o que permiti a reutilização do teste para validar se estes mesmos pontos do software continuam plenamente operacionais após manutenções futuras.

Frente a esse contexto, a automação de testes passa a ter um papel decisivo na manutenção dos softwares, pois torna ainda mais viável a realização dos testes funcionais de regressão (MOLINARI, 2008). O reflexo positivo é direto, com ganhos no tempo de realização dos testes e na qualidade do produto. Isso se deve principalmente ao fato de que a automação diminui o esforço de pessoal com os testes de regressão, bem como as partes críticas do produto de software passam a ser permanentemente exercitadas a cada ciclo de testes automáticos (MOLINARI, 2010; RIOS; MOREIRA, 2006).

Ainda que o capítulo atual demonstre a viabilidade da automação de testes, esta também enfrenta limitações que merecem atenção, o que será exposto no subcapítulo seguinte.

4.1 DIFICULDADES DA AUTOMAÇÃO DE TESTES DE SOFTWARE

Apesar de se mostrar um recurso que traz ganhos de performance à equipe de testes e de qualidade ao produto, a automação apresenta algumas dificuldades que merecem especial atenção de quem decidi adotá-la (MOLINARI, 2010).

A seleção adequada dos casos de testes a automatizar representa uma delas. Segundo Fewster e Graham (1999, tradução nossa) os casos de testes escolhidos para automação devem ser descritos como:

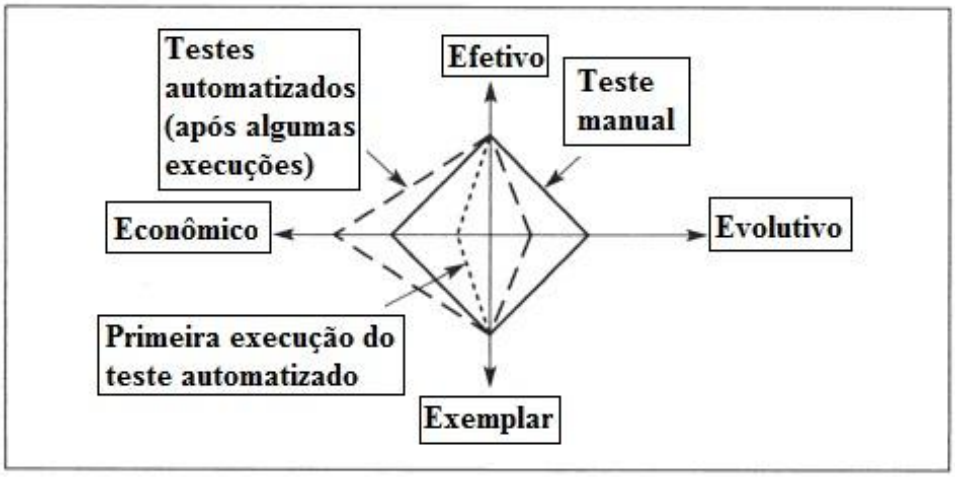
a) **econômicos:** precisam representar ganho econômico na sua realização, análise e correção;
b) **efetivos:** produzem como resultado a detecção de defeitos reais no software;

c) **evolutivos:** podem ser evoluídos de forma prática, à medida que o software também sofrer manutenções ou o próprio processo de automação de testes;

d) **exemplares:** possibilitam o teste de mais de um recurso do software, quantas vezes for necessário.

A figura 8 expõe um diagrama com o crescimento no número dos casos submetidos a testes, quando comparados os testes manuais com a execução automatizada dos mesmos. Tal resultado será obtido quando a automação for aplicada com a seleção de casos que contemplem as características já apontadas anteriormente. (FEWSTER; GRAHAM, 1999, tradução nossa).

Figura 8 – Diagrama da evolução dos testes manuais X automatizados.



Fonte: Adaptado de Fewster e Graham (1999, tradução nossa).

Além da dificuldade na identificação dos melhores casos de testes, a automação enfrenta outras limitações, sendo que entre as mais críticas merecem destaque:

a) **ir além ou aquém das necessidades reais:** quando um teste é realizado além do que precisa, gera perda de tempo na equipe de testes, nesse caso envolvendo também a equipe de automação de testes. Quando realizado aquém deixa em aberto a possibilidade de um defeito ser entregue. Isso torna obrigatório um alinhamento entre a estratégia de automação de testes com as reais necessidades de teste e o contexto do produto de software a ser testado (BACH; KANER; PETTICHORD, 2002, tradução nossa);

b) **ignorar que ferramentas de automação falham:** podem ocorrer falhas de gravação, falhas de reprodução, bem como falhas envolvendo a integração entre a ferramenta de automação e os demais recursos envolvidos. O uso

frequente aliado a constante estudo sobre alternativas aplicáveis diante de falhas é a saída possível e mais viável. Para circunstâncias extremas pode-se optar pela substituição da ferramenta, o que normalmente é mais trabalhoso e produz custo elevado (BACH; KANER; PETTICHORD, 2002, tradução nossa);

c) **implantar um processo de software para a automação:** assim como o produto de software que será submetido a testes pela automação, toda a produção de ferramentas de apoio, geração dos casos de testes e gravação de *scripts* são softwares, portanto devem contemplar as mesmas atividades do ciclo de vida de um processo de software (BACH; KANER; PETTICHORD, 2002, tradução nossa). A adoção de tarefas como projeto, desenvolvimento, testes, execução e manutenção devem ser empregados para dar segurança aos resultados que a automação produzirá (RIOS; MOREIRA, 2006);

d) **adoção de projetos para os *scripts:*** cada *script* equivale a um código programado de um software. Portanto requer os mesmos cuidados em sua estruturação, com emprego de projetos e testes que validem seu resultado (BACH; KANER; PETTICHORD, 2002, tradução nossa).

Apesar das limitações possíveis, existem outros fatores que favorecem o sucesso na adoção da automação de testes, os quais serão detalhados no próximo subcapítulo.

4.2 FATORES DE SUCESSO DA AUTOMAÇÃO DE TESTES DE SOFTWARE

Alguns fatores podem contribuir para que um projeto de automação de testes seja implantado com uma maior segurança de que os objetivos podem ser alcançados. Entre eles se destacam:

a) **planejar para depois implantar:** primeiro é preciso planejar, estudar, identificar as melhores ferramentas, assim como os cenários de uso e os recursos mais adequados, para só então partir para a implantação. Inverter esta ordem representa risco e segui-la amplia em muito as chances do êxito final (BACH; KANER; PETTICHORD, 2002, tradução nossa; MOLINARI, 2010);

b) **esperar para lucrar no momento certo:** como a automação requer implantação de todo um aparato e ainda dependerá da criação dos *scripts* de testes, só na segunda execução é que iniciarão os ganhos reais de performance

e custos. Evitar economia nas tarefas iniciais ajudará a garantir o sucesso (BACH; KANER; PETTICHORD, 2002, tradução nossa);

c) **melhor aproveitamento dos recursos:** é importante que o gestor da automação de testes esteja preparado para fazer o melhor uso possível dos recursos de estruturas e de pessoal. A automação de testes proporciona isso de forma natural, pois faz parte do próprio objetivo de um projeto de automação, de tal forma que melhorar esse aproveitamento é um dos fatores que contribuem para o sucesso do processo (MOLINARI, 2010)**;**

d) **aumento na quantidade de testes:** a consequência direta em se melhor aproveitar os recursos é uma ampliação na quantidade de testes que passa a ser realizado em um mesmo espaço de tempo em que antes eram praticados apenas os testes manuais. A figura 8 do subcapítulo anterior já expressava essa tendência, que naturalmente presta contribuição importante para o êxito do projeto (MOLINARI, 2010);

e) **aumento na quantidade de soluções de problemas entregues:** de maneira encadeada, à medida que aumentam os testes, é ampliada a quantidade de defeitos que são ajustados e entregues corrigidos ao cliente. O impacto na qualidade e satisfação do cliente final cresce, resultando em mais um fator que é decisivo na obtenção de sucesso para o projeto (MOLINARI, 2010);

f) **seleção adequada da ferramenta de automação:** a busca pela ferramenta ideal deve consumir algum tempo de pesquisa, sendo importante considerar três fatores para a tomada de decisão, os quais contribuirão de maneira decisiva para o êxito do projeto (BACH; KANER; PETTICHORD, 2002, tradução nossa),

- compatibilidade, pois o software de automação dos casos de testes deve ser executável nos sistemas operacionais e estruturas físicas dos equipamentos disponíveis para a tarefa de automatizar os testes (BACH; KANER; PETTICHORD, 2002, tradução nossa; RIOS; MOREIRA, 2006),
- familiaridade de interface, uma vez que deve se tratar de um software com interface amigável e intuitiva, que favoreça o uso mesmo por membros recém-contratados para a equipe de automação (BACH; KANER; PETTICHORD, 2002, tradução nossa),

- serviços de suporte do fabricante, devido ao fato de que é importante uma avaliação sobre os serviços oferecidos pela empresa desenvolvedora do software (BACH; KANER; PETTICHORD, 2002, tradução nossa). Uma vez escolhido um produto que não oferece um aparato de suporte, todo o trabalho de uma equipe de automação de testes será posto em risco caso surja uma pane em momento crítico, como por exemplo, diante de um prazo final de testes se esgotando;

g) **atenção às mudanças de interface no produto de software:** a equipe de automação precisa estar alinhada aos projetos futuros, conhecer as manutenções ou criações que estão por vir e antecipar ao máximo os ajustes em *scripts* de testes ou na própria ferramenta de automação. De forma resumida, as alterações de interface são frequentes e devem ser absorvidas pela equipe de automação de maneira positiva, o que também trará contribuição importante ao sucesso final do projeto (BACH; KANER; PETTICHORD, 2002, tradução nossa);

h) **evitar lógicas complexas:** a simplificação das lógicas empregadas na estruturação de *scripts* evita manutenções onerosas da própria automação dos casos de testes. A regra é simplificar e perseguir o êxito do projeto (BACH; KANER; PETTICHORD, 2002, tradução nossa);

i) **contratações coerentes com os objetivos e conhecimentos necessários:** contratar alguém com conhecimentos de programação, ferramentas de bancos de dados, noções de testes ou automação de testes é o caminho mais adequado na busca do sucesso para o projeto (MOLINARI, 2010; RIOS; MOREIRA, 2006);

j) **nunca automatizar no fim:** a automação deve ser iniciada nas primeiras fases do projeto, garantindo maior tempo de ação à equipe de automação, consequentemente ampliando as fases de estudos, projeto e desenvolvimento da automação. O resultado natural será uma ampliação nas chances de sucesso do projeto de automação em curso (MOLINARI, 2010);

k) **visão de que automatizar é investimento:** o uso da visão correta de que a automação é um investimento de médio ou longo prazo ajuda a evitar a incompreensão quanto a demora na obtenção dos resultados. Isso possibilita uma visão mais compreensiva desde o alto escalão até às equipes de testes

envolvidas no projeto, tornando ainda mais possível o alcance do sucesso desejado (MOLINARI, 2010);

l) **a adoção de métricas:** por último e não menos importante, gerenciar sem metas é algo praticamente impensável ou impraticável. Os responsáveis pela automação de testes devem adotar o mais cedo possível o uso de métricas de qualidade e de quantidade de testes (MOLINARI, 2010). Isso dará direção ao processo como um todo. As equipes atuarão com um rumo bem definido e os ajustes serão praticados mais facilmente, o que também tornará o êxito do projeto em algo mais alcançável.

Todos os fatores expostos acima dependem de um passo inicial de extrema importância, a decisão de que a automação de testes se tornou necessária.

Esse entendimento deve surgir a partir da visualização de que existe um ambiente maduro de testes manuais e que a estrutura já existente chegou ao limite daquilo que pode produzir em termos de resultados positivos. A partir desse ponto entra em cena a necessidade de se fazer mais, com o mesmo tempo disponível, porém com a inserção de um novo projeto. Tomar a iniciativa de adotar um projeto de testes automatizados com as cautelas apontadas anteriormente certamente direcionará ao êxito final que se pretenda atingir (MOLINARI, 2010).

Conforme exposto anteriormente, a tomada dessa decisão, entre outros fatores, requer o conhecimento das ferramentas mais adequadas ao projeto. Tendo isso em vista, o capítulo seguinte explorará algumas das ferramentas de automação para testes funcionais disponíveis.

5 FERRAMENTAS PARA AUTOMAÇÃO DE TESTES FUNCIONAIS

A implantação de um projeto de automação de testes requer o emprego de ferramentas que apoiem o processo. Entre as ferramentas dedicadas à gravação e execução de automações de testes disponíveis no mercado, podem ser destacadas as seguintes:

a) **Selenium:** ferramenta *open source* que é destinada à automação de testes funcionais em softwares para o ambiente *web* (MOLINARI, 2010). Compreende o software para gravação e execução dos *scripts*, ferramenta para controle remoto de páginas *web*, software que permite comandar um navegador *web* nativo localmente ou à distância e uma ferramenta que executa testes em diversos servidores *web* simultaneamente (SELENIUM, 2012, tradução nossa). Maiores detalhes sobre essa ferramenta serão estudados no subcapítulo seguinte;

b) **Autoit:** ferramenta *freeware* que permite a automação de testes funcionais em ambientes *desktop*, permitindo a iteratividade do usuário por meio de periféricos de entrada como o mouse e o teclado, comportando ainda a manipulação na entrada de dados, manipulação de janelas, interatividade com todos os controles padrões de janelas, entre outros recursos. Sua estrutura comporta uma documentação de exemplos e manuais, a própria ferramenta para gravação e execução dos *scripts* e um software que converte os *scripts* em arquivos executáveis (AUTOIT, 2012, tradução nossa).

c) **Testcomplete:** ferramenta comercial para testes funcionais de softwares em plataforma *desktop* e *web*. Possui ferramentas para gravação e execução dos *scripts*, ferramenta para permitir a execução de testes automatizados em máquinas que não dispõem do Testcomplete instalado e permite a leitura dos objetos de um browser para facilitar a automação, entre outros recursos (SMARTBEAR, 2012, tradução nossa).

O subcapítulo seguinte abordará características da ferramenta Selenium IDE, a qual será empregada na elaboração do trabalho atual.

5.1 A FERRAMENTA SELENIUM IDE

A suíte de testes é considerada como um grupo de ferramentas que viabilizam a automação de testes, sendo que entre elas podem existir desde softwares para automação da gerência e do planejamento de testes até aqueles destinados à gravação e execução da automação de testes (RIOS; MOREIRA, 2006).

A ferramenta de automação denominada de Selenium IDE compreende um software de automação dos casos de testes, responsável pela gravação e execução dos *scripts* de automação direcionados a páginas da *web* (SELENIUM, 2012, tradução nossa). Para efeito de maior entendimento, é importante destacar que a ferramenta Testlink pode ser empregada para automatizar a gerência e o planejamento dos testes (TESTLINK, 2012, tradução nossa), a qual será referenciada em abordagens posteriores.

O software Selenium *Integrated Development Environment* (IDE) funciona de forma acoplada ao navegador Mozilla Firefox, atuando como um *plug-in* (MOLINARI, 2010). Entre seus recursos disponíveis podem ser destacados (SELENIUM, 2012, tradução nossa):

a) a seleção inteligente de campos do software, o que facilita a gravação e reexecução dos testes;
b) o uso dos recursos de auto completar os comandos mais comuns do Selenium, facilitando a codificação de *scripts*, quando necessário;
c) o suporte à depuração dos *scripts*, com possibilidade de estabelecimento de pontos de parada que auxiliam nos ajustes de *scripts*;
d) a variedade de *plug-ins* que permite expandir os recursos da IDE para facilitar tarefas específicas de automação, como por exemplo, a inserção de uma *toolbar* que facilite o acesso à IDE a partir do navegador;
e) a geração do conteúdo gravado em formato *Hypertext Markup Language* (HTML), facilitando a reexecução da automação;
f) suporta o controle por algumas linguagens de programação, como por exemplo, C#, Java, Pearl, *Hypertext Preprocessor* (PHP), Python, Ruby, entre outras;
g) pode ser executado nos sistemas operacionais OS, Microsoft Windows, OS X, Linux, Solaris, entre outros;
h) recurso atenuador do problema de perda de sincronismo entre o *script* e o software automatizado, também conhecido como calibragem;

A limitação mais visível do Selenium é a sua dependência de execução que limita os testes ao browser Mozilla Firefox das versões 4 a 10 (SELENIUM, 2012, tradução nossa).

O uso da ferramenta ocorre com a realização de etapas que compreendem desde a baixa do software de automação de testes funcionais *web* até sua execução e avaliação de resultados. Os passos básicos para gravação e execução da automação de testes com o Selenium IDE são (SELENIUM, 2012, tradução nossa):

a) **acionamento da ferramenta para gravação de *scripts*:** uma vez instalado o Selenium IDE, ao abrir o produto de software *web* a ser testado e acionar no navegador o menu Ferramentas, opção Selenium, a janela de gravação de *scripts* será aberta, já em estado de gravação das atividades em curso. Sempre que for necessário interromper ou retomar a gravação, o usuário deverá utilizar a opção de menu Ação e logo em seguida Gravar, ou ainda, utilizar o botão de gravação no canto superior esquerdo do Selenium IDE. Após gravar cada caso de teste, o mesmo deve receber uma denominação em suas propriedades, estas acessíveis pelo menu sensível ao contexto de cada caso de teste, ou seja, pelo botão inverso do *mouse*. O caso de teste criado deverá ser salvo em formato HTML, caso haja interesse em reutilizá-lo. Após a criação de diversos casos de testes, uma suíte de teste poderá ser salva, o que permite o agrupamento de casos de testes conforme as dependências, funcionalidades ou rotinas em que se aplicam;

b) **execução dos testes:** com o suporte de um software para automação da gerência e planejamento de testes, como por exemplo o software Testlink (TESTLINK, 2012, tradução nossa), ou ainda, com a seleção manual diretamente no Selenium IDE, respeitando uma ordem de execução predefinida por meio de um plano de testes, com uso de planilhas;

c) **manutenção dos *scripts*:** após a gravação do *script* de teste e salvamento em formato HTML, será possível editá-lo, o que também é um facilitador às tarefas de manutenção da própria automação de testes, sendo possível alterar, inserir ou apagar comandos, além de ser permitido mudanças no fluxo de execução do software e nos conteúdos que serão gravados.

Ao fim da sequência de testes automatizados a ferramenta Selenium IDE cumprirá a tarefa de apresentação dos resultados obtidos para cada caso de teste executado, apontando falhas e mensagens de erros, caso ocorram. Sem que haja prejuízo didático, o trabalho atual

emprega os próprios recursos de detecção de erros e falhas que o Selenium IDE dispõe. Contudo é necessário considerar que nem todas as falhas detectadas pela ferramenta Selenium IDE representam necessariamente falhas ou erros no software em teste, uma vez que estes podem ter sua origem no próprio *script* em execução. Assim convém ressaltar que na prática a análise do resultado de testes pode ser realizada com o suporte de um software exclusivo para a gerência da automação, normalmente comparando dados ou *screenshots* entre versões distintas do software testado (SELENIUM, 2012, tradução nossa).

6 TRABALHOS CORRELATOS

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

6.1 DESENVOLVIMENTO DE UMA METODOLOGIA APLICADA AO GERENCIAMENTO E ACOMPANHAMENTO DE TESTE DE SOFTWARE VIA WEB

Trabalho proposto por Fernando Bettiol Lopes para conclusão da graduação em Ciência da Computação pela Universidade do Extremo Sul Catarinense-UNESC em 2009.

O principal objetivo foi o desenvolvimento de uma ferramenta para gerência dos testes de software pela técnica funcional ou caixa-preta, denominado de *I&T Manager* (*Issue and Test Manager)*. O estudo comportou a busca de conhecimento sobre engenharia de software, compreensão da técnica de testes funcionais, aplicação dos conceitos da engenharia de software e testes caixa-preta para elaborar a ferramenta, bem como o acompanhamento do ciclo de vida do teste de software rastreando os erros funcionais e por fim a aplicação da ferramenta automática para gerenciamento dos testes.

O software *I&T Manager* foi desenvolvido com uso da linguagem Java, facilitando o acesso em diversos sistemas operacionais bem como via *web*. Também foi elaborado para ser operado em modo multiusuário, com cada usuário tendo perfil diferenciado e compatível com a função que exerce no uso do *I&T Manager*. O software gerencial comporta as seguintes rotinas:

a) **planejamento do teste:** possibilitado pelo cadastro ou manutenção dos procedimentos, casos, atividades e ciclo de vida do teste. Também pode ser cadastrada uma ferramenta de *debug*;
b) **execução do teste:** possível por meio do cadastro ou manutenção do cronograma, bem como a seleção dos casos, programa a ser testado, ferramentas a serem empregadas e documentação dos resultados do teste;
c) **análise dos resultados do teste:** viabilizado por relatórios dos resultados do teste, números das métricas do teste, assim como visualização da fluência nas etapas de teste para determinado software;

d) **recursos de debug:** o gerenciador de testes comporta o cadastro de erros conhecidos para integração com ferramenta de *debug*, conhecida como *issue track*, como por exemplo, o software Bugzilla.

Os relatórios gerenciais de horas de teste por programa, erros detectados por usuário e erros detectados por software testado, aliados às parametrizações e cadastros, segundo Lopes (2009), permitem que o software atenda a praticamente todas as carências de um ambiente de testes, trazendo como resultado uma melhora no desempenho produtivo da equipe que o empregar.

6.2 AUTOTEC: UMA FERRAMENTA DE AUTOMAÇÃO DE TESTES PARA INTERFACES DINÂMICAS

Apresentado em 2010 por Ismael München, o trabalho foi um dos requisitos para obtenção do título de Especialista em Computação Aplicada, o qual foi concedido pelo Centro de Ciências Tecnológicas-CCT do Departamento de Ciência da Computação na Universidade do Estado de Santa Catarina-UDESC (MÜNCHEN, 2010).

Focado em dinamizar a automação de testes funcionais regressivos, buscando assim aprimorar a qualidade dos softwares que tenham constantes manutenções na interface, o trabalho teve como principal objetivo o desenvolvimento de um software para automação de casos de testes, denominado Autotec. Tal software atua reconhecendo as alterações de interface no produto de software testado, ajustando automaticamente os *scripts* de teste, eliminando assim as manutenções manuais nos *scripts* de testes automatizados.

O software foi desenvolvido para atuar de forma compatível com a ferramenta *Computer-Aided Software/System Engineering* (CASE) TOTVS Metadados, a qual é empregada no desenvolvimento de softwares e faz o armazenamento das informações sobre construções de interfaces em seu banco de dados. Isso permite ao Autotec acesso direto a qualquer mudança ocorrida em um produto de software desenvolvido por meio de tal ferramenta CASE.

Ensaios apontados no trabalho indicam que o Autotec detectou automaticamente as alterações de interface no produto de software quanto à ordem de campos, retirada de campos, campo habilitado ou não e inclusão de campos, sem a necessidade de se modificar *scripts* dos casos de testes. O reconhecimento das modificações na interface permite que tomadas de decisões sejam automatizadas frente ao *script* de teste, como por exemplo, a

indução de valor padrão para um determinado campo novo, com um tipo de dado já conhecido. Supondo que este campo novo exija um valor específico, ainda é possível a apresentação de uma interface ao usuário da automação, requisitando a inserção do valor pertinente.

Tais estudos apontaram como resultado o fato de que softwares de automação que atuam de forma dinâmica a exemplo do Autotec, tornam ainda mais viável a aplicação da automação de testes funcionais regressivos, justamente por dispensarem qualquer intervenção manual para geração ou manutenção dos *scripts*, trazendo produtividade ao processo de teste, à medida que automatizar se torna mais rápido, bem como agregando qualidade no produto de software entregue, pelo aumento na quantia de funcionalidades testadas de forma regressiva.

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

Artigo elaborado pelos pesquisadores Adalberto Nobiato Crespo, Odair Jacinto da Silva, Carlos Alberto Borges, Clênio Figueiredo Salviano, Miguel de Teive, Argollo Junior e Mario Jino, com apoio das instituições Ampla Consultoria em Informação, Centro de Pesquisas Renato Archer (CenPRA) e UNICAMP, todas da cidade de Campinas. Também apoiou o trabalho a Universidade de São Francisco, da cidade de Itatiba.

O principal objetivo do trabalho é apresentar uma metodologia que facilite a adoção, implantação e uso de processos de testes nas empresas desenvolvedoras de software. Para isso empregou-se um estudo sobre teste e qualidade de software, com análise da norma IEEE 829 (CRESPO et al, 2012; IEEE, 2008, tradução nossa), a qual determina padrões e normatizações para a implantação do processo de testes de software, abrangendo o gerenciamento, aquisição, desenvolvimento, operação e manutenção de um processo de testes de software, tudo isso subordinado ao conceito de um plano de teste mestre (IEEE, 2008, tradução nossa).

A partir desse estudo, com base na norma IEEE 829, foi elaborada uma metodologia adaptável à realidade de qualquer produto de software ou empresa desenvolvedora. Tal metodologia foi estruturada pelo CenPRA, buscando a simplificação para facilitar assim a adesão das entidades desenvolvedoras. Nesse contexto de simplificação a metodologia proposta passou a conter os seguintes componentes:

a) **treinamento:** comporta a capacitação de equipes de testes apresentando-lhes conceitos e técnicas pertinentes à tarefa, sendo possível definir as necessidades diferenciadas de cada equipe ou empresa;

b) **processo de teste:** abrange um arcabouço de processo de teste, comportando o planejamento, projeto, execução e monitoramento dos testes. Poderão ser empregadas as fases de testes de unidade, integração, sistemas e aceitação. Partindo do arcabouço de processo de teste cada empresa desenvolvedora deverá definir suas necessidades e prioridades;

c) **suporte para a geração de documentos:** constitui-se de um método para gerar documentos que serão empregados para gerenciar a rotina dos testes, tanto na preparação de dados e casos de testes quanto na documentação dos resultados obtidos. Originalmente a metodologia está fundamentada na norma IEEE 829-1998 que comporta 8 documentos que regem o planejamento, especificação e registro dos testes.

Segundo os autores, a proposta é que com a adoção dessa metodologia, ocorram ganhos não só na identificação e correção de defeitos, mas também em se prevenir falhas futuras no produto de software.

Um experimento foi realizado pela equipe, implantando a metodologia do CenPRA em uma pequena empresa desenvolvedora de software na cidade de Campinas. A escolha da metodologia se deu pela possibilidade da mesma suportar diversos modelos de avaliação, inclusive a ISO/IEC TR 15504 (ISO, 1998, tradução nossa), a qual foi a opção adotada por permitir o controle de níveis de capacidade para o processo de teste.

O resultado final obtido pela microempresa foi a obtenção do nível 2 de capacidade em teste de software, com ganhos diretos na melhoria da qualidade do produto, do relacionamento com clientes e no desempenho das equipes envolvidas no processo de software. Segundo os autores também ficou claro que a metodologia do CenPRA possibilitou uma transparência quanto ao relacionamento da equipe de desenvolvimento com a equipe de teste, gerando uma sinergia importante para o andamento de projetos futuros.

6.4 TESTE DE SOFTWARE AUTOMATIZADO: UMA SOLUÇÃO PARA MAXIMIZAR A COBERTURA DO PLANO DE TESTE E AUMENTAR A CONFIABILIDADE E QUALIDADE EM USO DO SOFTWARE

O artigo foi proposto em 2010 pelos autores Marcantonio Catelani, Lorenzo Ciani, Valeria L. Scarano e Alessandro Bacioccola, os quais integram a equipe do Departamento de Eletrônica e Telecomunicação da Universidade de Florença, na Itália.

Resumidamente o trabalho apresenta uma metodologia que permite a criação automatizada e acelerada de casos de testes funcionais regressivos, bem como testes funcionais focados na detecção de erros relacionados à ocorrência de *memory leak* (vazamento de memória).

Aplicando uma mistura de parâmetros de entrada, provenientes do histórico de testes anteriores, bem como a distribuição estatística dos dados para os campos do produto de software, a metodologia aplica uma geração dinâmica e pseudorrandômica dos casos de testes. O resultado é que a combinação das possíveis variáveis e consequentemente a evolução do estado de cada uma, representam os próprios casos de testes gerados automaticamente.

Analisando o montante de memória física disponível no computador, a metodologia proposta também pode submeter o produto de software à execução exaustiva, tentando assim expor as falhas de desenvolvimento que levariam a um vazamento de memória. Um exemplo clássico desse defeito é a criação e não eliminação de componentes que ocupam muitos Bytes de memória.

O estudo aponta que a adoção da metodologia proposta permite ganhos consideráveis no número de casos de testes regressivos executados por cada hora de teste realizado. A tabela 3 a seguir detalha tais informações:

Tabela 3 – Comparativo entre Testes Manuais x Testes Automatizados x Testes Automatizados Acelerados.

Dias	Horas Humanas	Horas de Máquina	Testes Manuais	Testes Automatizados	Testes Automatizados Acelerados
1	8	24	40	336	840
7	56	168	280	2352	5880
15	120	360	600	5.040	12.600

Fonte: Adaptado de Catelani et al (2010, tradução nossa).

Analisando a tabela 3, ganha destaque a diferença entre o total de testes automatizados acelerados, proposto pelo artigo, frente à automação de testes tradicional para cada hora de trabalho.

Os ensaios realizados pelos autores apontaram que tanto os testes regressivos realizados de forma mais acelerada, quanto os testes em busca de vazamento de memória, contribuíram decisivamente para o aumento da qualidade do software em uso, ampliando a satisfação do cliente pela confiabilidade e permitindo ainda uma redução nos custos de manutenção do produto de software.

7 AUTOMAÇÃO DE TESTES REGRESSIVOS NA AMPLIAÇÃO DA COBERTURA FUNCIONAL DE SOFTWARES COM MANUTENÇÕES FREQUENTES

A tentativa de garantir a qualidade no produto de software invariavelmente passa pela necessidade de submetê-lo a testes (MYERS, 2004, tradução nossa), inclusive funcionais (PRESSMAN, 2006). Da mesma forma também é importante considerar que os produtos de software que recebem manutenções constantes são mais propensos a apresentar defeitos (SOMMERVILLE, 2003).

Essa combinação de fatores torna importante um estudo com base no teste de regressão, o qual é um tipo de teste funcional voltado a garantir que requisitos funcionais antes em pleno funcionamento, permaneçam igualmente operantes após uma manutenção (PRESSMAN, 2006).

O custo para aplicar testes de regressão de forma exaustiva é alto quando praticado manualmente, principalmente pelo tempo que será consumido (MYERS, 2004, tradução nossa), tornando a tarefa quase que impraticável de maneira manual.

O presente trabalho foi realizado com uso da automação de testes funcionais na fase de regressão, buscando assim o reaproveitamento de casos que tenham sido validados em testes manuais de versões ou compilações anteriores de um determinado produto de software.

O processo da automação de testes foi praticado com uso da ferramenta Selenium IDE (SELENIUM, 2012, tradução nossa), sendo que a seleção dos casos de testes a automatizar ocorreu por meio de critério de derivação não formal, baseado no conhecimento e experiência do especialista de testes.

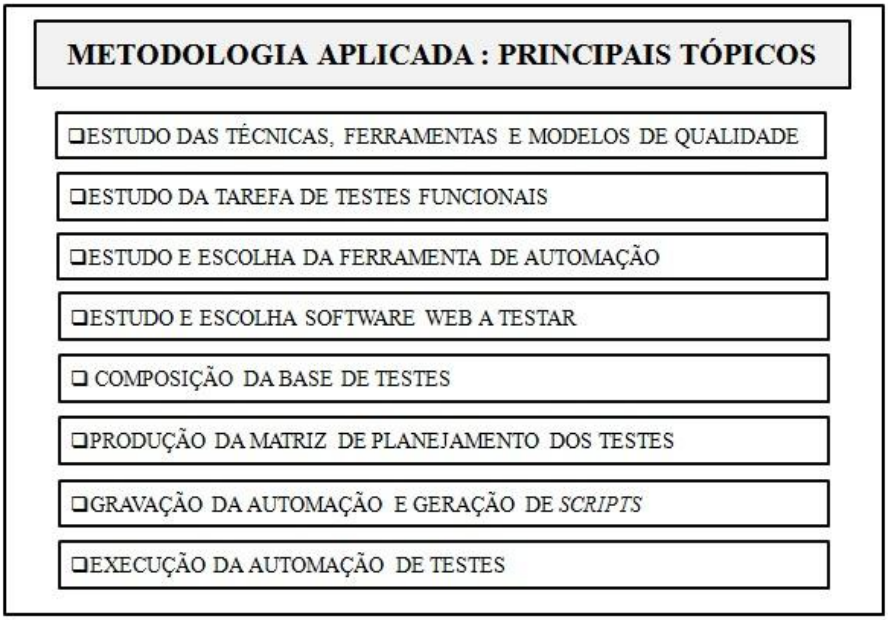
O presente trabalho foi elaborado conforme consta na figura 9, a qual demonstra esquematicamente as principais etapas que serão abordadas nos próximos subcapítulos.

7.1 METODOLOGIA

A metodologia aplicada para a elaboração do trabalho foi distribuída em 10 etapas, contemplando desde o levantamento de bibliografias, passando por estudos realizados sobre as informações levantadas, bem como pela aplicação de técnicas, métodos e ferramentas pesquisadas.

O esquema da figura 9 demonstra sucintamente a metodologia, porém é válida uma análise completa da mesma, avaliando todas as fases, o que poderá ser apreciado nos subcapítulos seguintes.

Figura 9 – Esquema da metodologia aplicada.



Fonte: O próprio autor (2012).

7.1.1 Levantamento Bibliográfico

Tomando como base inicial o apoio e as orientações recebidas, foram realizadas pesquisas em diversas bibliotecas, incluindo as bibliotecas da UNESC, PUC/POA, UFSC/Florianópolis, estas visitadas localmente, além de diversas outras via web.

Também foram realizadas pesquisas nas bases de dados *Science Direct e Scielo* por meio da UNESC, bem como foram pesquisados periódicos e publicações acadêmicas da PUC/POA. A prática de pesquisas em aberto via ferramentas de busca da internet também ocorreu.

Em todas as pesquisas houve a preocupação em se confirmar a fonte geradora da informação, o destino de aplicação, sua contextualização e coesão com os objetivos do trabalho bem como, e até mais importante que os demais pontos, qual a relevância científica que cada material possuía, como por exemplo, anais em que haviam sido publicados, ano de publicação, trabalhos posteriores que foram originados a partir destes, entre outros fatores.

Nas pesquisas realizadas via web, em sites de bibliotecas ou nas bases de dados, ocorreram inferências de termos e palavras chave ligadas aos objetivos do trabalho, com uso

de múltiplas variações e combinações para termos compostos, tudo isso nas línguas portuguesa e inglesa, sempre visando a ampliação da lista de resultados obtida a cada busca.

Quanto aos livros em inglês, foram realizadas traduções via ferramenta web, com posterior correção gramatical e de semântica, empregando para isso conhecimentos em língua inglesa, bem como de termos técnicos da área de computação, além de pesquisas em dicionários ou publicações voltadas à língua inglesa.

No decorrer da pesquisa bibliográfica chamou a atenção o grande volume de publicações acadêmicas surgidas nos últimos 5 anos em outras regiões e que são voltados para a área de testes de software, tanto no âmbito de artigos, quanto em dissertações de graduação, teses de mestrado e doutorado.

Entre os materiais pesquisados tiveram grande impacto no resultado do trabalho os livros dos autores Pressman (2006), Sommerville (2003), Molinari (2010), Molinari (2008) e Myers (2004, tradução nossa), principalmente pelo embasamento teórico obtido quanto à engenharia de software, qualidade de software, testes de software e automação de testes de software. É justo observar que os demais autores trouxeram também contribuições importantes, complementando ou expandindo em diversos momentos as ideias e teorias trazidas pelos autores já citados.

Na linha das pesquisas realizadas em artigos, dissertações e teses houve a confirmação de diversas teorias pregadas pelos autores supracitados, trazendo para estas publicações um caráter complementar por meio da visão acadêmica sobre tais teorias. Também é necessário frisar que tais publicações acadêmicas proporcionaram relevante ganho quanto às técnicas de redação e estruturação textual do trabalho.

Quanto aos trabalhos correlatos, foi perceptível o fato de que existem no Brasil diversos trabalhos que se equiparam a produções internacionais, a tal ponto que foi custoso detectar um trabalho estrangeiro que tivesse algo diferenciado de outros que haviam sido localizados nacionalmente.

Entre todos os aspectos positivos que a pesquisa bibliográfica permitiu, é importante ressaltar que além do impacto mais óbvio da pesquisa, que é a contribuição para o andamento e conclusão do presente trabalho, houve um importante ganho acadêmico proporcionado pela sua vivência, já que o ato de pesquisar constitui um dos pilares de todas as ciências, inclusive da computação. Outros impactos relevantes foram percebidos no âmbito profissional, pelos conhecimentos adquiridos e perfeitamente aplicáveis, assim como no contexto pessoal, pela satisfação no envolvimento com o processo de pesquisa como um todo.

Dando sequência à metodologia, no subcapítulo seguinte serão abordados temas sobre estudos que empregaram as bibliografias levantadas.

7.1.2 Estudo Sobre as Técnicas, Ferramentas e Modelos de Qualidade da Engenharia de Software

Tomando como base os livros de diversos autores, como por exemplo, Ghezzi, Jazayeri e Mandrioli (1991, tradução nossa), Guerra e Colombo (2009), Ledgard (1987, tradução nossa), Peters e Pedrycz (2000, tradução nossa), Rocha, Maldonado e Weber (2001), Pressman (2006) e Sommerville (2003), entre outros, foram estudados diversos temas inerentes à engenharia de software, abordando desde modelos, ferramentas e técnicas para o gerenciamento de projetos, passando pela importância e necessidade de estudos quanto à qualidade de software, sem perder a atenção para softwares que enfrentam manutenções frequentes.

Nesse âmbito foi observado que quanto maiores são as intervenções em um software, igualmente crescem as chances da ocorrência de erros (PRESSMAN, 2006), ao passo que segundo Ghezzi, Jazayeri e Mandrioli (1991, tradução nossa), cerca de 60% de todo o esforço e custos de produção das equipes ocorrem devido a tarefas ligadas à manutenção do software.

Tais estudos apontaram que softwares de frequentes manutenções requerem uma atenção diferenciada das equipes de produção, principalmente devido ao fato de que a manutenabilidade está entre os mais importantes fatores da qualidade produtiva do software, sendo esta composta principalmente pela facilidade dos reparos e a praticidade da evolução do software (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa).

As pesquisas realizadas quanto à qualidade de software também apontaram a importância em se conhecer as normatizações de produção e modelos de qualidade existentes.

Para essa finalidade foram realizados estudos das normas CMMI (SEI, 2010, tradução nossa) na versão 1.3, MPS.br (SOFTEX, 2011) e NBR ISO/IEC 9126 (ABNT, 2003), o que trouxe um maior entendimento sobre o enfoque que tais normas aplicam quanto à manutenabilidade e a tarefa de testes de software, validando a importância dos testes na busca da qualidade, bem como denotando a necessidade de seu emprego da forma mais eficaz e produtiva possível.

Os estudos também revelaram os esforços já existentes nesse sentido, como por exemplo, o Programa Brasileiro de Qualidade e Produtividade em Software (PBQP) (BRASIL, 2006; GUERRA; COLOMBO, 2009) mantido por diversas entidades de ensino, órgãos governamentais e entidades privadas, o qual visa incentivar as melhores práticas na busca da qualidade e produtividade do software Brasileiro e que já apoiou diversos projetos de pesquisa sobre qualidade com uso de testes de software.

Aliando o conhecimento adquirido acerca da engenharia de software, qualidade de software e teste de software, ao o fato de que, segundo Sommerville (2003), a tarefa de testes pode consumir cerca de 50% do tempo de produção para uma versão de determinado software, tornou-se importante um estudo mais minucioso sobre a tarefa de testes, suas principais técnicas, métodos e fases, o que será abordado minuciosamente no próximo subcapítulo.

7.1.3 Estudo Sobre a Tarefa de Teste Caixa-preta

Embasado nas publicações de diversos autores, foram realizados estudos buscando os conceitos do teste de software, metodologias, técnicas e suas fases, bem como as respectivas aplicações para a tarefa de produção de software com qualidade.

Entre as técnicas de testes ganhou destaque a caixa-preta ou funcional, justamente por atuar com enfoque na validação dos requisitos funcionais do produto (PRESSMAN, 2006). Contudo a pesquisa pôde demonstrar que a adoção do teste funcional tem como um de seus maiores desafios o produção desordenada de documentações relacionadas aos requisitos funcionais, o que pode ser minimizado por meio da adoção de modelos de desenvolvimento ou padrões adequados de produção.

As pesquisas também demonstraram a divisão de fases existente para esta técnica, sendo que a fase de regressão foi a opção aplicável para o trabalho, posto que a automação de testes requer a execução dinâmica do software, tal qual ocorre na fase de sistema em que o software é executado para a devida avaliação. Até mesmo por essa característica, foi observado que alguns autores defendem que a fase de regressão é complementar e análoga à fase de sistemas, já que ambas ocorrem via execução dinâmica do software.

Conforme Pezzè e Young (2008), o diferencial da fase de regressão está em possuir grande impacto para softwares de manutenção frequente, visto que seu intuito é

assegurar que casos de testes validados manualmente em versões anteriores, sigam válidos na versão que estiver em testes.

Frente aos conhecimentos adquiridos se pôde observar que a prática do teste caixa-preta, quando empregado na fase de regressão, coopera para o aumento da qualidade do software, pois mantém a aderência do mesmo a seus requisitos funcionais originais, justamente pela ampliação da cobertura funcional que é proporcionada, o que é um dos principais objetivos do trabalho. Tendo como base tais informações, mostrou-se importante um estudo sobre os critérios de derivação de casos de teste caixa-preta, o que será o tema abordado no próximo subcapítulo.

7.1.4 Estudo e Seleção do Critério de Derivação dos Casos de Testes

A análise dos critérios de derivação mais relevantes recebeu importante contribuição da bibliografia de diversos autores, entre eles Delamaro, Maldonado e Jino (2007), Molinari (2008), Myers (2004, tradução nossa), Pressman (2006) e Rios e Moreira (2006). Baseado nos estudos foi possível compreender o conceito, possíveis aplicações e até as limitações dos critérios formais e não formais de derivação dos casos de teste caixa-preta.

A pesquisa permitiu observar a importância da aplicação dos testes funcionais com vistas a minimizar os prejuízos quanto a prazos e a qualidade, ocasionados por testes além ou aquém do necessário (RIOS; MOREIRA, 2006; ROCHA, MALDONADO; WEBER, 2001).

Igualmente foi relevante constatar que testes funcionais e regressivos são importantes, porém impraticáveis manualmente de forma massiva, abrangendo todas as variações possíveis de testes (MYERS, 2004, tradução nossa).

Os estudos apontaram que critérios formais de derivação, baseados em particionamento de equivalências, análise do valor limite, grafo causa-efeito e matriz ortogonal, entre outros, todos voltados à técnica caixa-preta, podem ser empregados de forma complementar entre si, sendo importante a utilização de dois ou mais critérios formais combinados para a obtenção dos casos de testes que resultem em mais eficiência. Da mesma forma foi possível constatar que a adoção de algum critério de derivação, quer seja ele formal ou não formal, é fator crucial para a obtenção de testes com maior abrangência funcional e maior qualidade no produto de software.

Frente a isso é pertinente observar que segundo Rios e Moreira (2006), o uso do critério não formal baseado no conhecimento e experiência do especialista, pode ser empregado na busca de uma maior qualidade dos testes e do produto.

Após a apreciação das características entre as opções disponíveis, o critério não formal com base no conhecimento e experiência do especialista foi o escolhido para a realização do trabalho, considerando que na região de Criciúma este é o critério de derivação comumente empregado, visando assim a obtenção de resultados mais reais, que sirvam de base sólida para a comparação com o resultado de futuros estudos acadêmicos, que envolvam os critérios formais de derivação.

A aplicação do critério de derivação escolhido se deu nas fases de composição da base de testes e da matriz de planejamento dos testes, as quais serão abordadas mais adiante. O subcapítulo seguinte tratará dos estudos e experimentos praticados durante a escolha do software de automação a ser empregado no trabalho.

7.1.5 Estudo e Seleção da Ferramenta de Automação dos Testes Funcionais:

Tomando como referência inicial o auxílio e as orientações recebidas, foram realizados diversas pesquisas em sites e livros correlatos, entre os quais merecem destaque o livro de Molinari (2010) e as documentações disponíveis nos sites das ferramentas Autoit (AUTOIT, 2012, tradução nossa), Selenium IDE (SELENIUM, 2012, tradução nossa), e Testcomplete (SMARTBEAR, 2012, tradução nossa).

Os estudos demonstraram as características de cada ferramenta, sendo que todas possuíam o suporte à gravação e execução de *scripts* de automação.

A ferramenta Testcomplete se mostrou como a única opção comercializada, o que justifica sua maior robustez e atuação em testes com ambientes *desktop* e *web.*

Entre as outras duas ferramentas, ambas *open source*, as pesquisas demonstraram que o software Autoit possui as funções necessárias para uma prática adequada e segura de testes funcionais em ambientes *desktop,* sendo que a ferramenta Selenium IDE possui características assemelhadas no que se refere às funcionalidades básicas para automação de testes, porém ela é focada exclusivamente em testes funcionais de softwares em ambiente *web.*

Considerando o intento didático do trabalho, a aquisição do software Testcomplete se mostrou desnecessária, ao passo que tanto o Autoit quanto o Selenium IDE poderiam cumprir os objetivos almejados.

Entre as duas ferramentas, a escolha pelo Selenium IDE se deu pelo interesse científico em conhecer e manusear este software posto que o mesmo é destinado para o ambiente *web*. Além do interesse científico, testes demonstraram a praticidade de uso e a segurança da ferramenta Selenium IDE, por exemplo, durante a fase de estudos e escolha do software a ser testado para realização do trabalho, tópico este que será abordado no próximo subcapítulo.

7.1.6 Estudo e Seleção do Software *Web* a Testar

Os estudos iniciais da ferramenta Selenium IDE permitiram observar incompatibilidades com testes funcionais de softwares desenvolvidos com algumas tecnologias, como por exemplo, PHP, dotNet, entre outras. Como exemplo disso pode-se citar as falhas de *scripts* observadas com o texto "Element link=modelo not found" para o site "ead.unesc.com.br", na sessão de download dos materiais de aula. Falhas semelhantes foram observadas em outros softwares testados durante o estudo, contudo todos tinham características iguais ligadas à não localização de elementos ou componentes dos sites.

Apesar de alguns estudos realizados, não foi confirmado um meio de solução para as falhas. Ainda assim é válido afirmar que o Selenium IDE possua recursos que atenuem tais falhas de ausências de campos ou não localização de referências, principalmente devido à sua robustez e diversidade de recursos, contudo diante da importância quanto ao prosseguimento do trabalho, a escolha do software a ser submetido aos testes funcionais também passou a ter como critério a confirmação sobre qual plataforma de desenvolvimento poderia ser usada sem apresentar tais críticas. Assim, foram realizados alguns experimentos com softwares desenvolvidos puramente com a linguagem Java, os quais não apresentaram as falhas observadas anteriormente.

Após essa constatação inicial, o software avaliado foi o Domínio Atendimento, desenvolvido pela empresa Domínio Sistemas Ltda, a qual possui sede na cidade de Criciúma.

A autorização para realização dos estudos foi obtida junto à Diretoria Técnica da empresa, que disponibilizou não somente o software em si, como também o ambiente oficial

de execução, com acesso via usuários supervisores devidamente cadastrados e preparados para os testes.

O software Domínio Atendimento é desenvolvido com a linguagem Java e baseado em interface gráfica, sendo que a gravação e execução dos *scripts* de automação ocorreram sem falhas, confirmando os experimentos iniciais com outros softwares dessa linguagem. É pertinente ressaltar que o Atendimento foi um dos softwares pioneiros em seu gênero no Brasil, pois dá suporte ao relacionamento entre escritórios de contabilidade com seus respectivos clientes, sem que isso represente um custo adicional para os escritórios, já que o mesmo é considerado comercialmente como um dos módulos do sistema Domínio Contábil Plus, um dos softwares para escritórios contábeis mais conhecidos e difundidos no Brasil.

O principal recurso do Atendimento está na opção que permite o registro de solicitações de serviços (SS), estas registradas por clientes ou pelo próprio escritório. Tal recurso permite que sejam solicitados serviços, como por exemplo, a emissão de contratos, gerações de declarações ao fisco, geração de arquivos informativos fiscais, emissão de livros contábeis, além de possibilitar a solicitação de cálculos de folhas de pagamento, férias e rescisões, entre outros diversos recursos disponíveis. Como informação complementar, é pertinente destacar que após uma SS ser registrada pelo cliente, os usuários do escritório que estiverem operando o Domínio Contábil Plus, o qual é desenvolvido para uso em ambiente *Desktop*, recebem de forma automática e online uma notificação sobre a SS registrada, podendo assim, diretamente dentro do software Contábil, realizar as tarefas requisitadas pelo cliente e no mesmo ambiente, já reportar a solução do que foi requisitado.

Apesar da aparente baixa complexidade do software Atendimento, este possui características específicas quanto às permissões de usuários, que podem ser de tipo supervisor de escritório ou de cliente, ou ainda um usuário comum com permissões para determinadas rotinas ou acesso a determinados departamentos, clientes, além de outras restrições de acesso. Também é possível observar uma considerável complexidade na própria sequência de situações em que uma SS pode se encontrar, passando desde a situação “Sem análise” após seu registro, como pelas situações “Em análise escritório”, esta disponível apenas para usuários de escritórios, e as situações “Aguardando resposta”, “Respondido” e “Concluída”.

Quando referenciados os termos situação da SS ou alteração da SS, deve-se considerar que as equipes de produção, usuários de escritórios e de seus clientes chamam cada alteração de situação de uma SS pelo nome de “trâmite”. Assim, é necessário considerar que

as funcionalidades de cadastro ou alteração de SS passam por trâmites que correspondem às situações já citadas acima.

Outra particularidade interessante do software é que ao fim de um atendimento, quando concluído pelo cliente, o mesmo pode votar quanto à satisfação que obteve para a SS concluída, podendo escolher entre Satisfeito, Muito Satisfeito, Insatisfeito, Muito Insatisfeito ou Desejo Não Opinar.

Tendo como base as votações realizadas, o software disponibiliza um gráfico que proporciona uma oportunidade única para o escritório de contabilidade avaliar o desempenho de sua equipe quanto ao atendimento prestado aos clientes.

Conforme já citado, apesar de haver uma simplicidade aparente, o software tem suas complexidades, ao ponto que para efeito de estudos, a geração dos casos de testes que compuseram a base de testes, tema do subcapítulo a seguir, ocorreu buscando a melhor combinação para apenas algumas das funcionalidades disponíveis, justamente para que a finalidade acadêmica dos experimentos fosse mantida.

7.1.7 Composição da Base de Testes

A execução da etapa ocorreu com vistas a levantar os casos de testes mais adequados, que cobrissem as funcionalidades mais críticas, levando em conta para isso a frequência de manutenções recebidas, bem como o uso constante da rotina, medindo assim a importância em se verificar o funcionamento permanente de tais rotinas.

Com esse intuito foram delimitadas as funcionalidades a testar, visando a eficiência dos casos de testes e consequente ganho de qualidade. Nesse sentido foram empregados conhecimentos adquiridos em etapas anteriores sobre a qualidade de software, testes funcionais regressivos, assim como foi utilizado o critério de derivação com base no conhecimento e experiência do especialista quanto ao software Domínio Atendimento.

A tabela 4 demonstra as suítes de testes geradas, as principais funcionalidades contempladas na base de testes e o total de casos de testes para cada suíte.

Analisando a tabela 4, merece destaque a suíte de testes “3,0-CadastrandoeTramitandoSS”, a qual contempla o maior número de casos de testes, justamente por comportar as opções de cadastros (válidos e tentativas inválidas), tramitações, visualizações e conclusões de SS. É importante observar que este é um dos requisitos

funcionais mais utilizados por escritórios e clientes, sendo também um dos que mais recebem manutenções.

Tabela 4 – Base de testes sumarizada.

Suíte De Teste	Funcionalidades abrangidas	Número de Casos de Testes
0,0-TestandoAcessos	Acesso negado e acesso permitido (com usuários Supervisor de Escritório e Supervisor de Cliente)	6
1,0-CadastrandoUsuarios	Cadastro válido, tentativa de cadastro inválido (usuários supervisor de Cliente e supervisor de Escritório)	4
2,0-CadastrandoeExcluindoNoticia	Cadastro válido (notícia para funcionários e notícia para clientes), tentativa de cadastro inválido, visualização pelo cliente e exclusão de notícia.	7
3,0-CadastrandoeTramitandoSS	Cadastro pelo Escritório, trâmites Em Análise, Em Análise Escritório, Aguardando Resposta, Respondido (Cliente), tentativa de visualizar trâmite bloqueado (Cliente), Concluído (Escritório), cadastro pelo Cliente, Concluído pelo cliente (com votações de Satisfeito, Muito Satisfeito, Insatisfeito, Muito Insatisfeito e Não Desejo Opinar).	27
999,0-CasosdeTestesUsoGeral	Fechamento de sessão, acesso com usuários Supervisores (Cliente e Escritório)	3

Fonte: O próprio autor (2012).

A escolha da cobertura de funcionalidades como o acesso negado (ou permitido), cadastro de usuários e cadastro de notícias levou em conta o fato de que, apesar de se tratar de rotinas que recebem pouca manutenção, um erro ou falha em qualquer uma destas representa uma ocorrência crítica, podendo impedir acessos, bloquear envio de notícias importantes ou mesmo negar o cadastro e adesão de novos usuários. Com o intento de gerar casos de testes para uma automação funcional regressiva que garanta o permanente funcionamento de tais atividades, houve a necessidade de se registrar pelo menos um caso de teste para estas funcionalidades, novamente empregando o critério de derivação baseado no conhecimento e experiência do especialista de testes, bem como os conhecimentos teóricos adquiridos em etapas anteriores.

Aplicando o mesmo critério foi possível constatar que para os casos de testes envolvendo funcionalidades das SS, como por exemplo, cadastros e alterações de situação, alguns requisitos funcionais não precisavam ser contemplados por completo, com todas as suas variações de valores possíveis, sendo que outros poderiam ser combinadas entre si, o que

contribuiu para a redução do número de casos de testes, frente as variações totais que seriam possíveis caso todas as combinações fossem cobertas. Uma análise da matriz de planejamento dos testes permitirá uma melhor compreensão, o que será exposto no subcapítulo posterior.

7.1.8 Composição da Matriz de Planejamento dos Testes

A aplicação das orientações recebidas, aliadas ao conhecimento e experiência do especialista, permitiu a elaboração da matriz de planejamento dos testes, contemplando colunas condizentes com diversos requisitos funcionais que necessitariam de pelo menos um caso de teste contemplando-os, para assim ocorrer uma validação mais adequada do software frente a possíveis manutenções por meio da ampliação da cobertura funcional.

Os requisitos listados em colunas receberam os nomes visíveis de forma esquemática na figura 10.

Quanto à disposição das colunas na matriz, vale considerar que estão dispostas da esquerda para a direita respeitando a mesma ordem visível na figura de cima para baixo. É importante frisar que a matriz também recebeu a coluna Sistema_Operacional como sendo a última, a qual não corresponde a um requisito funcional, mas que pode, ainda que em raras exceções, ser empregada para determinar a cobertura de testes funcionais para os sistemas operacionais Windows 7 ou Linux, posto que os softwares *web* e o Selenium IDE têm a característica de operar em modo multiplataforma.

Quanto aos requisitos funcionais apresentados, estes permitem que o especialista de testes, por meio dos filtros dinâmicos da planilha, consiga delimitar o raio de ação dos testes funcionais regressivos, sejam eles manuais ou automatizados, selecionando somente casos de testes que sejam compatíveis com as funcionalidades que passarão por uma manutenção, que sofrerão impactos conhecidos devido a experiências anteriores ou que se pretenda submeter a testes para se estabelecer um bom nível de segurança em pontos mais críticos do software ou diante de situação crítica exigindo ação mais ostensiva e abrangente.

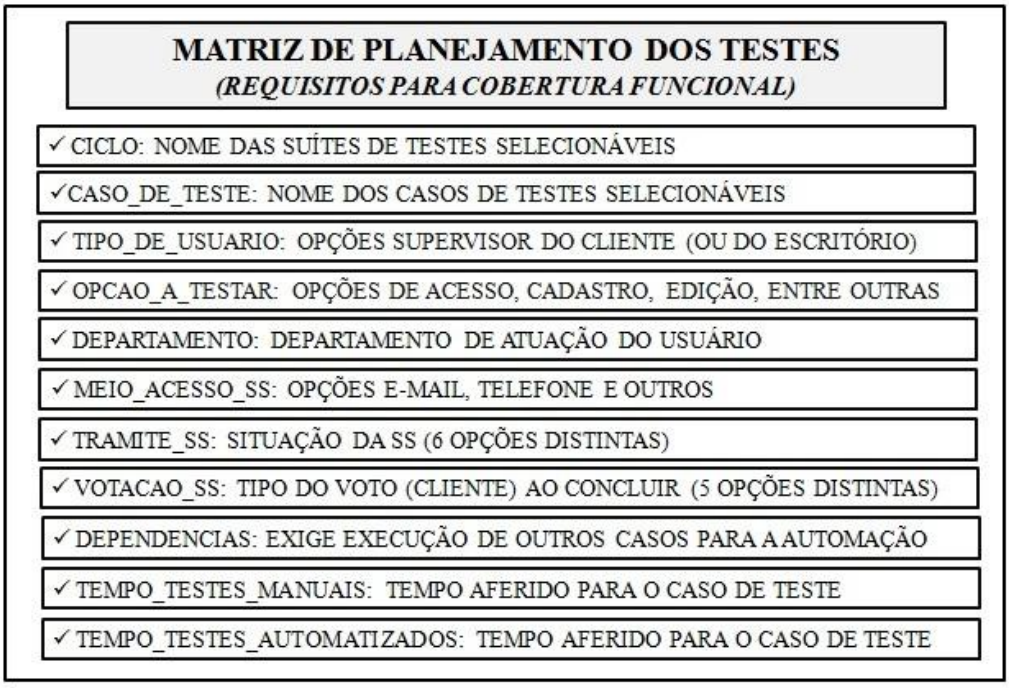
Para algumas das colunas foram determinadas informações de identificação dos casos de testes automatizados, como por exemplo, as colunas Suite e Casos_deTestes.

O detalhamento sobre a forma de criação das suítes e casos de testes será exposto mais adiante, em outra etapa da metodologia, abordada no próximo subcapítulo.

Também merece destaque o fato de que a composição da matriz de planejamento permitiu observar algumas variações de testes que poderiam ser descartadas, como por

exemplo, as que foram cogitadas com base nos valores possíveis para a coluna Departamento, a qual comporta as opções Contabilidade, Pessoal, Fiscal, Patrimônio, Registro de Empresas, Financeiro e Administrativo. A análise do montante de casos de testes que seriam necessários para testar todas as combinações dos valores possíveis para o campo departamento, com os demais requisitos funcionais e seus respectivos valores possíveis, demonstrou que os objetivos didáticos do trabalho seriam onerados, assim, optou-se por manter tal requisito funcional na matriz, porém contemplando apenas a opção Contabilidade.

Figura 10 – Composição da matriz de planejamento dos testes.



Fonte: O próprio autor (2012).

É relevante considerar que as funcionalidades que sofreriam impacto direto pelo campo Departamento, envolvendo questões de permissões de uso para usuários que não são supervisores, também não foram contemplados nesse trabalho, pelos mesmos motivos de se evitar que os estudos fossem prejudicados pelo volume de casos de testes a abranger.

Outra coluna que merece uma menção especial é a Dependencias, a qual indica quais casos de teste precisam ser executados pelo menos uma vez antes do caso de teste selecionado ser posto em atividade em uma automação de testes. Essas dependências existem devido ao fato de que os estudos foram realizados em um ambiente de execução oficial do software Atendimento, o que impede que uma base de dados seja previamente preparada com cadastros mais elementares que apoiem os testes de uma versão do software em produção.

Frente a isso os casos de testes devem ser executados considerando uma ordem de precedência para que dados sejam cadastrados durante a execução dos testes, de forma manual ou automatizada. Contudo essa situação não gera prejuízo, uma vez que os estudos dos testes manuais versus automatizados ocorreram empregando o mesmo ambiente. Também é correto avaliar que a coluna Dependencias perde a importância se a seleção dos casos de testes ocorrer com base nas suítes de testes, as quais foram automatizadas na ordem correta de precedência, o que se caracteriza em mais um conhecimento específico do especialista de testes, que deverá empregá-lo ao executar os testes automatizados.

As colunas Tempo_Testes_Automatizados e Tempo_Testes_Manuais compreendem os tempos que foram aferidos durante as etapas de composição da base de testes e de gravação da automação de testes, tempos estes que servirão para nortear o especialista quanto ao tempo que certamente ocorrerá nas próximas execuções dos testes que forem selecionados. Os tempos coleados tiveram papel importante para os estudos do trabalho, o que será mais bem observado na análise dos resultados obtidos.

Considerando todas as características da matriz de planejamento dos testes observa-se que esta tende a cumprir com seu papel de prover uma maior e melhor cobertura dos testes, além de aumentar a acurácia nos resultados, favorecendo o emprego de testes funcionais regressivos automatizados, visto que o uso dos filtros dinâmicos, conforme já dito, permitem ao especialista a seleção de casos de testes mais aplicáveis frente às manutenções realizadas.

Uma vez estruturada a matriz de planejamento dos testes, o trabalho pôde evoluir para a próxima etapa, com a geração dos testes automatizados, o que será abordado no subcapítulo seguinte.

7.1.9 Execução dos Testes Manuais com Gravação da Automação e Criação dos *Scripts*

Empregando a ferramenta Selenium IDE, foram realizados os testes manuais que deram origem às suítes e casos de testes automatizados. Para isso a ferramenta foi acionada a partir do software Mozilla Firefox, com a execução pelo menu Ações e a opção Gravar do Selenium IDE, conforme já exposto anteriormente no trabalho.

Após acionada a gravação, ocorreu a execução dinâmica do software Domínio Atendimento, com acesso após informar os *logins* de usuário Supervisor de Escritório ou Supervisor de Cliente, aplicando manualmente os testes elencados na matriz de planejamento

dos testes, inclusive respeitando a ordem de precedência adequada diante das dependências dos casos de testes selecionados.

Na primeira execução dos testes para gravação da automação foi possível observar que o tempo de realização dos testes manuais, combinados com a gravação dos testes e salvamento dos *scripts* em HTML, consumiram cerca de 35% a mais do que o praticado nos testes manuais, quando realizados sem executar as tarefas de automação. Essa ocorrência reforçou o fato de que somente a partir da segunda execução de testes automatizados, é que surgem os ganhos reais, bem conforme Fewster e Graham (1999, tradução nossa) observam em sua bibliografia.

A criação das suítes de testes se deu conforme as denominações visíveis na tabela 4, exposta em subcapítulo anterior, sendo que para os casos de testes procurou-se a criação de nomes condizentes com a opção principal a ser executada. A tabela 5 apresenta a lista dos casos de testes que tiveram os *scripts* gerados e suas respectivas suites.

Visando uma melhor organização dos casos de testes, estes também foram codificados com uma classificação que identifica cada caso de forma única e ao mesmo tempo, o relaciona com a sua respectiva suíte de testes, o que também é possível observar na tabela 5, como por exemplo, o caso de teste denominado de "3,01-CadastrandoSSCompletaMeioAcessoTelefone.html", o qual pertence à suíte "3,00-CadastrandoeTramitandoSS". Assim, os códigos "3,00" e "3,01" representam um agrupamento que facilita a contagem de casos de testes para cada suíte, bem como a identificação durante o manuseio, evitando confusões e possíveis erros.

Por fim é válido observar que o processo de gravar e salvar as automações foi facilitado pela interface amigável e intuitiva do Selenium IDE, inclusive para manutenção dos *scripts*, possibilitando a alteração dos comandos quanto a seu objeto alvo, bem como o valor a ser gravado ou manipulado pelo comando. Após a conclusão das gravações, veio a etapa de execução dos *scripts*, adaptações e coleta de tempos, o que será tratado em detalhes no subcapítulo a seguir.

Tabela 5 – *Scripts* da base de testes composta

Casos de Testes	Suítes de Testes
0,01-UsuarioSupervisorClienteUsuarioInvalido.html	0,00-TestandoAcessos
0,02-UsuarioSupervisorClienteSenhaInvalida.html	0,00-TestandoAcessos
0,03-UsuarioSupervisorClienteUsuarioComSenhaValida.html	0,00-TestandoAcessos
0,04-UsuarioSupervisorEscritorioUsuarioInvalido.html	0,00-TestandoAcessos
0,05-UsuarioSupervisorEscritorioSenhaInvalida.html	0,00-TestandoAcessos
0,06-UsuarioSupervisorEscritorioUsuarioComSenhaValida.html	0,00-TestandoAcessos
1,01-CadastroUsuarioSupervisorCliente.html	1,00-CadastrandoUsuarios
1,02-Cadastro0UsuarioSupervisorClienteInvalido(sem dados).html	1,00-CadastrandoUsuarios
1,03-CadastrandoUsuarioSupervisorEscritorio.html	1,00-CadastrandoUsuarios
1,04-CadastroUsuarioSupervisorEscritorioInvalido(semdados).html	1,00-CadastrandoUsuarios
2,01-CadstrandoNoticiaValida.html	2,00-CadastrandoeExcluindoNoticias
2,02-EditandoNoticiaValida.html	2,00-CadastrandoeExcluindoNoticias
2,03-ApagandoNoticiaValida.html	2,00-CadastrandoeExcluindoNoticias
2,04-CadastrandoNoticiaInvalida.html	2,00-CadastrandoeExcluindoNoticias
2,05-CadastrandoNoticiaValidaParaCliente.html	2,00-CadastrandoeExcluindoNoticias
2,06-VisualizandoNoticiaCliente.html	2,00-CadastrandoeExcluindoNoticias
2,07-ApagandoNoticiaValidaParaCliente.html	2,00-CadastrandoeExcluindoNoticias
3,01-CadastrandoSSCompletaMeioAcessoTelefone.html	3,00-CadastrandoeTramitandoSS
3,02-CadastrandoSSInvalida.html	3,00-CadastrandoeTramitandoSS
3,03-CadastrandoSSCompletaMeioAcessoEmail.html	3,00-CadastrandoeTramitandoSS
3,04-CadastrandoSSCompletaMeioAcessoOutros.html	3,00-CadastrandoeTramitandoSS
3,05-AlterandoSSCompletaMeioAcessoTelefone-EmAnaliseEsc.html	3,00-CadastrandoeTramitandoSS
3,06-AlterandoSSCompletaMeioAcessoEmail-EmAnalise.html	3,00-CadastrandoeTramitandoSS
3,07-AlterandoSSCompletaMeioAcessoEmail-RespondidoEsc.html	3,00-CadastrandoeTramitandoSS
3,08-AlterandoSSCompletaMeioAcessoOutros-EmAnalise.html	3,00-CadastrandoeTramitandoSS
3,09-AlterandoSSCompletaMeioAcessoOutros-ConclPeloEsc.html	3,00-CadastrandoeTramitandoSS
3,10-QuestionandoSSCompletaMeioAcessoE-mail.html	3,00-CadastrandoeTramitandoSS
3,11-AnalisandoSSCompletaMeioAcessoTelefone.html	3,00-CadastrandoeTramitandoSS
3,12-ConcluindoSSCompletaMeioAcessoOutros-ConclEscritorio.html	3,00-CadastrandoeTramitandoSS
3,13-AlterandoSSCompletaMeioAcessoEmail-QuestionamClie.html	3,00-CadastrandoeTramitandoSS
3,14-ConcluindoSSCompletaMeioAcessoTelefone-ConclEscr.html	3,00-CadastrandoeTramitandoSS
3,15-ConcluindoSSCompletaMeioAcessoEmail-Satisfeito.html	3,00-CadastrandoeTramitandoSS
3,16-CadastrandoSSCompletaPeloCliente-MuitoSatisfeito.html	3,00-CadastrandoeTramitandoSS
3,17-CadastrandoSSCompletaPeloCliente-Insatisfeito.html	3,00-CadastrandoeTramitandoSS
3,18-CadastrandoSSCompletaPeloCliente-MuitoInsatisfeito.html	3,00-CadastrandoeTramitandoSS
3,19-CadastrandoSSCompletaPeloCliente-NãoOpinar.html	3,00-CadastrandoeTramitandoSS
3,20-AlterandoSSCompletaPeloEscritorio-MuitoSatisfeito.html	3,00-CadastrandoeTramitandoSS
3,21-AlterandoSSCompletaPeloEscritorio-Insatisfeito.html	3,00-CadastrandoeTramitandoSS
3,22-AlterandoSSCompletaPeloEscritorio-MuitoInsatisfeito.html	3,00-CadastrandoeTramitandoSS
3,23-AlterandoSSCompletaPeloEscritorio-NãoOpinar.html	3,00-CadastrandoeTramitandoSS
3,24-AlterandoSSCompletaPeloClie-ConclComMuitoSatisfeito.html	3,00-CadastrandoeTramitandoSS
3,25-AlterandoSSCompletaPeloClie-ConclComInsatisfeito.html	3,00-CadastrandoeTramitandoSS
3,26-AlterandoSSCompletaPeloClie-ConclComMuitoInsatisfeito.html	3,00-CadastrandoeTramitandoSS
3,27-AlterandoSSCompletaPeloClie-ConclComNaoOpinar.html	3,00-CadastrandoeTramitandoSS

Fonte: O próprio autor (2012).

7.1.10 Execução da Automação de Testes

Finalizada a gravação de todos os *scripts*, os mesmos foram postos em execução para avaliar e validar o funcionamento, visando entre outros objetivos, a correção e melhor adequação de dados ou comandos empregados originalmente.

A tarefa de execução das automações ocorreu por meio da abertura de cada suíte de teste, acionando em seguida o menu Ação e a opção Play do Selenium.

Após iniciada a execução dos scripts o Selenium IDE permite que o usuário opere o próprio Selenim IDE para visualização do andamento da execução, como também possibilita que outros softwares possam ser operados livremente, sem que a automação seja afetada.

Apesar de a bibliografia do Selenium IDE não deixar isso claro, por medida de cautela, houve o cuidado de não utilizar o Firefox enquanto as automações estavam em execução. Também por questões de se evitar interferências quanto aos tempos de execução, nos momentos em que houve a contagem de tempo, nenhum outro software foi posto em execução, buscando-se com isso a exatidão da contagem do tempo, sem o risco da intervenção de terceiros quanto ao uso de memória, processamento ou acesso a disco.

Quanto ao resultado dos testes, ao fim da execução de cada uma das suítes, havendo falha em algum caso de teste, este é demonstrado em cor vermelha na lista de casos de testes, contudo se a execução transcorrer bem, todos os casos estarão marcados em cor verde. Para as ocorrências de falhas, o Selenium permite uma visualização do erro detectado, conforme é possível observar na figura 11, coletada diante de uma falha induzida na execução da automação.

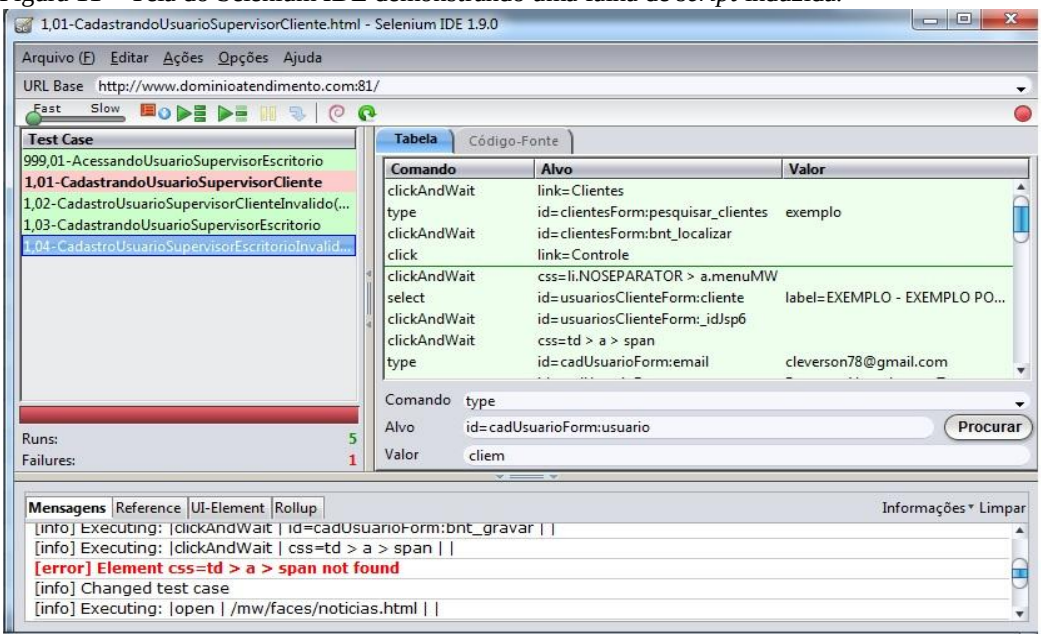
Frente a estas questões de erros, mereceu destaque o fato de que o campo "Usuário" do cadastro de SS chegou a apresentar uma falha de *script* devido a um *delay* do software Atendimento, quanto à montagem da lista de usuários após a seleção de conteúdo no campo Cliente, o que impediu o Selenium de escolher o usuário "Cleverson Usuário de Cliente", usuário este escolhido para ser gravado nos casos de testes cadastrais de SS.

Em testes manuais observou-se que ao informar o nome do Cliente e pressionar rapidamente a tecla tab para avançar de campo e novamente para sair do campo Usuário, a lista também não é montada e a SS é gravada sem a informação do usuário, confirmando que não se trata de uma falha do *script*, mas sim um comportamento adverso do software Atendimento pelo tempo de espera para listagem de usuários do Cliente.

Considerando que o Selenium IDE possui opção para calibragem do tempo de execução dos *scripts* e do tempo máximo de espera quando determinado componente permanecer sem apresentar resposta para a automação, tentou-se contornar o caso ajustando estas opções.

A velocidade de execução do Selenium varia do nível rápido ao lento, com 5 estágios de velocidade entre um e outro, sendo que ao abrir a ferramenta, esta já vem pré-configurada como velocidade rápida. Já quanto ao tempo de espera sem resposta, o Selenium já é aberto configurado com o tempo de 30.000 milissegundos.

Figura 11 – Tela do Selenium IDE demonstrando uma falha de *script* induzida.



Fonte: O próprio autor (2012).

As tentativas de resolver o problema do campo Usuário no cadastro da SS inicialmente giraram em torno de configurações de velocidade menor e maior, bem como do tempo de espera, combinando diversas variantes possíveis, contudo além da permanência do erro quanto ao campo Usuário, observou-se uma demora extrema na execução, sendo que também surgiram novas ocorrências de erros, inclusive em outros scripts que antes operavam normalmente.

Diante do que foi observado, partiu-se para outra solução, tentando alterar o comando "select" do Selenium, no qual o erro ocorreu, sendo que em pesquisas na documentação da própria IDE e empregando o *plugin* Firebug na análise dos componentes do site Atendimento, assim como fazendo ajustes e submetendo a novas execuções, foi possível constatar que o comando "typeKeys" do Selenium interrompeu a ocorrência da falha, permitindo assim a execução do *script*.

Apesar do nome de usuário ter permanecido sem ser gravado nas execuções da automação após os ajustes, o estudo da automação visando ampliar a cobertura funcional na fase de regressão não sofreu qualquer prejuízo, posto que todas as demais funcionalidades

permaneceram sendo executadas e validadas, além de ser importante considerar que a versão oficial do produto, quando testado manualmente, também apresenta o mesmo comportamento.

A execução dos casos de testes automatizados permitiu a observância de diversas características que dão destaque para o uso da automação de testes funcionais regressivos na ampliação da cobertura funcional, o que será o tema central do subcapítulo a seguir.

7.2 RESULTADOS OBTIDOS

Embasado pela bibliografia pesquisada bem como em experiências anteriores envolvendo a automação de testes em ambiente *desktop*, havia a expectativa de que as primeiras execuções da automação de testes funcionais *web* trariam benefícios em relação a qualidade do produto testado.

Após as primeiras tomadas de tempo verificou-se que a automação proporcionaria vantagem frente à execução dos testes de maneira manual.

O tempo de testes praticado manualmente para as suítes e casos de testes documentados na base, sem as rotinas de gravação da automação, foi aferido em 30 minutos e 41 segundos (1841 segundos).

Já a primeira execução das mesmas suítes e casos de testes na tarefa de gravação dos testes e geração dos *scripts* consumiu cerca de 35% a mais do que o tempo consumido manualmente, perfazendo um tempo aproximado de 40 minutos, justamente devido às tarefas de geração da automação e salvamento de *scripts* que estavam envolvidas.

No momento em que a automação de testes foi posta em execução, ocorreu uma redução do tempo de testes para um total de 8 minutos e 03 segundos (483 segundos).

A tabela 6 detalha os resultados iniciais obtidos na comparação entre a execução manual versus a execução automatizada dos testes, para cada suíte de testes. É correto observar que a suíte de testes “999,0-CasosdeTestesUsoGeral”, visível na tabela 4 do subcapítulo 7.1.7 que trata da composição da base de testes, está com seu tempo diluído entre as demais suítes, justamente porque possui uma função de complementação para os demais casos de testes automatizados.

A análise da tabela 6 fica mais clara quando consideradas as seguintes características dos casos de testes contemplados nas suítes:

a) os dados de testes da suíte “0,0-TestandoAcessos”, mesmo sendo simples, demonstram com maior clareza o quanto a automação pode promover em

termos de ganho de performance e melhor aproveitamento do tempo das equipes de testes, dado que automatizar uma rotina simples não requer tanto esforço e no longo prazo, pela relevância crítica da opção, haverão ganhos importantes;

b) para a suíte de testes "1,0-CadastrandoUsuários" houve um ganho importante de tempo, se comparado à anterior. Isso se deve principalmente porque esta já é uma suíte em que os testes, propositalmente, foram realizados tanto de forma manual, quanto de forma automatizada, com digitações de dados em um nível moderado. Isso denota o fato de que, quanto maior o volume de digitação envolvida nos testes, maiores serão os ganhos da automação;

c) quanto à suíte de testes "2,0-CadastrandoeExcluindoNotícia", para esta fica ainda mais evidente o ganho exponencial de tempo de teste com a prática automatizada frente a manual, justamente porque as notícias possuem uma base puramente textual, sendo que nessa suíte foram aplicados textos mais amplos do que na suíte anterior;

d) na suíte "3,0-CadastrandoeTramitandoSS" a percepção de que o volume textual é fator diferencial para o ganho de tempo nos testes automatizados se consolida, visto que nesta suíte ocorre uma queda no ganho de tempo, justamente a suíte em que ocorrem maiores seleções de campos baseados em listas de informações e as digitações de textos ocorrem somente em 2 campos.

Tabela 6 – Testes manuais X automatizados.

Suítes de Testes	Tempo dos Testes Manuais	Tempo dos Testes Automatizados	Ganho % com Testes Automatizados
0,0-TestandoAcessos	00:01:42	00:00:30	240%
1,0-CadastrandoUsuarios	00:02:33	00:00:36	325%
2,0-CadastrandoeExcluindoNoticia	00:03:45	00:00:41	449%
3,0-CadastrandoeTramitandoSS	00:22:41	00:06:16	262%

Fonte: O próprio autor (2012).

Buscando minimizar impactos de agentes externos, como por exemplo, a concorrência no processamento, acesso a disco ou memória em servidores, bem como o tráfego de rede, foram realizados testes com a automação na madrugada de 17/10/2012, entre as 6h e 07h03min, sendo que foram realizadas novas coletas de tempo, as quais estão documentadas na tabela 7.

Considerando que houve um ganho maior para a suíte "3,0-CadastrandoeTramitandoSS", a mesma passou por 3 execuções com cronometragem dos

tempos, sendo a primeira com tempo de 04min e 26s, a segunda com 04min e 21s e na terceira execução o tempo já havia subido para 04min e 42s, sendo que esta já iniciou próximo das 07h e se encerrou pouco após esse horário.

Tabela 7 – Testes manuais X automatizados (fora do horário comercial).

Suítes de Testes	Tempo dos Testes Manuais	Tempo dos Testes Automatizados	Ganho % com Testes Automatizados
0,0-TestandoAcessos	00:01:42	00:00:25	308%
1,0-CadastrandoUsuarios	00:02:33	00:00:36	325%
2,0-CadastrandoeExcluindoNoticia	00:03:45	00:00:39	477%
3,0-CadastrandoeTramitandoSS	00:22:41	00:04:21	422%

Fonte: O próprio autor (2012).

Os testes adicionais, com cronometragem do tempo de execução da automação fora do horário comercial, se mostraram importantes pelos seguintes fatores:

a) demonstraram que após a implantação de um processo formal e gerenciado de automação de testes, que comporte a execução das automações de forma agendada para ocorrer nas madrugadas, os ganhos em termos de otimização dos processos serão ainda maiores;

b) também ficou provado que fatores externos interferem no tempo de execução da automação, principalmente em processos mais críticos e de uso mais intenso, como o caso dos cadastros e tramitações de SS, largamente utilizados por clientes da Domínio Sistemas, o que certamente produz concorrência de processamento, acesso a disco e gerenciamento de memória nos servidores. Contudo não se pode descartar o fator do tráfego de rede, que mesmo com os testes tendo sido realizados em um ambiente de rede estável, com velocidade sendo constantemente aferida, diversos agentes externos envolvendo estruturas de rede podem ter contribuído para o aumento do tempo de execução em horário comercial;

c) também foi notório o fato de que as funcionalidades de acesso, mas principalmente as de cadastro e exclusão de notícias, também tiveram um ganho considerável na execução durante a madrugada, mesmo estas sendo tarefas mais simples, o que pode representar um sinal de que fatores envolvendo tráfego de rede tenham uma relação maior com os tempos computados nas execuções em horários comerciais;

d) por fim também é necessário considerar que como as funcionalidades de cadastro de usuários mantiveram exatamente o mesmo tempo na coleta de testes dentro e fora do horário comercial, sendo que estes são recursos que geram pouca concorrência entre usuários, pode-se considerar tais fatores reforçam a ideia de que o verdadeiro ponto de impacto tenha sido a concorrência entre usuários ao acessar recursos largamente utilizados, como por exemplo, as telas de listagens de SS ou os formulários de cadastro de SS.

O fato é que a discrepância de tempos observada deixa em aberto estudos que possam cooperar para uma execução mais otimizada das automações, seja melhorando as execuções em horário comercial, seja se programando para que as execuções ocorram de madrugada, sempre que possível.

Retomando o foco central do trabalho e considerando que o tempo total de execução de todas as suítes, quando comparada à execução manual versus automatizada em horário comercial, constata-se um ganho na casa dos 281% com a adoção da automação dos testes.

Isso equivale a dizer que enquanto um testador atuar manualmente em todas as suítes da base de testes, a automação terá sido rodada praticamente 3 vezes, ou ainda, que quando o testador chegar próximo a 1/3 de seus testes manuais, a automação já terá se encerrado.

Apenas essa constatação já é suficiente para confirmar que o uso da automação de testes precisa se tornar uma realidade para as empresas de software, visto que permite, com um menor esforço da equipe, ampliar a capacidade produtiva e a cobertura funcional, isso em um espaço de tempo no qual seria realizado um menor volume de testes, se praticados de forma manual.

Segundo Molinari (2012), é certo que a necessidade de manutenção dos scripts, bem como manter uma equipe de automação bem treinada e destinada apenas a tarefas de automação consomem tempo e custo, contudo os ganhos a longo prazo são notórios e irrefutáveis.

O uso da automação de testes funcionais, aplicando uma base de testes sólida, moldada com critérios de derivação que visam produtividade e garantia de qualidade, tendem a permitir que o teste de regressão seja aplicado aproveitando ainda mais a capacidade de ganho produtivo que a própria automação proporciona.

Um exemplo hipotético disso seria considerar que o software Atendimento passará por uma alteração relacionada ao cadastro e tramitações de SS quando esta é registrada pelo Escritório, passando a existir uma inversão na sequência dos campos Usuário e Departamento, sendo que o Departamento passa a vir antes do Usuário. Nesse cenário os *scripts* de automação não precisarão sofrer alteração, visto que a seleção de campos ocorre de forma dinâmica, o que permite considerar apenas o tempo de execução na comparação entre os tempos de testes manuais e automatizados.

Considerando ainda um cenário um pouco mais crítico, em que uma correção foi executada em um ponto específico de cada uma das demais suítes de testes, uma no cadastro de usuários de clientes, eliminando uma mensagem de erro que ocorria ao gravar o cadastro e outra no cadastro de notícias, em que foi retirada uma falha que causava a perda de textos digitados, mas que não eram salvos pelo Atendimento quando o usuário acionava a opção de gravação da notícia.

Como visto todos os testes acima são relativamente simples e pontuais, ocorre porém que o teste manual pode vir a ocorrer de forma induzida, seja por alguma orientação repassada ao testador, seja por conhecimento prévio de ocorrências, seja pela boa intenção de se fazer mais e melhor. É certo que dependendo da experiência e conhecimento do testador, os casos de testes estruturadas poderão atingir bons ou até excelentes níveis de garantia, mas isso deixa exposto o risco de que uma falha só será percebida pelo cliente, quando for operar o software, daí a importância em se planejar os testes de forma criteriosa, empregando os conhecimentos e experiência do especialista de testes, com uso de ferramentas apropriadas, como por exemplo, a matriz de planejamento dos testes.

Piorando um pouco o cenário, supondo que esta é a terceira versão de manutenção do software em 5 dias e que dessa vez tudo tem que dar certo, sendo que são 17h20min e em 30 minutos, após vir para testes, a versão terá que ser liberada devido a um prazo contratual que um cliente precisa cumprir até as 18h, o qual depende exclusivamente do software Atendimento para realizar tal tarefa.

A análise desse caso hipotético deve ser realizada considerando que a matriz de planejamento de testes do trabalho, supostamente contempla todas as funcionalidades do Domínio Atendimento.

Nesse cenário já se sabe que testar todas as suítes manualmente não será possível, pois o tempo para a realização de tal tarefa será de 30 minutos, ao passo que selecionar casos de testes de forma aleatória poderá permitir que a situação do cliente se complique pelo

surgimento de um novo erro, em um ponto crítico que estava funcionando antes dessa versão, e este só ser percebido pelo cliente, o que seria catastrófico.

Nesse contexto a aplicação dos conhecimentos e experiência do especialista de testes, aliadas a uma bem estruturada base de testes, esta implantada em uma matriz de planejamento de testes, permitirá ao especialista a geração de filtros dos casos de testes a executar, com segurança, aplicando em sua análise os seguintes critérios:

a) pelo menos um caso de teste deve contemplar o registro de uma nova SS pelo Escritório;
b) a SS registrada deve passar pelos trâmites de Em análise, Em análise Escritório, Aguardando Resposta do cliente, Respondida pelo cliente, novamente Aguardando Resposta do Cliente e Concluída pelo cliente com votação uma única vez, para assim serem testadas todas as variações de trâmites pelo menos uma vez;
c) devido ao nível crítico da rotina, deverá haver também ao menos um caso de teste que contemple o registro de SS por parte do cliente, rotina que não deve ser afetada pela alteração principal, mas que deve ser testada para garantir a continuidade do funcionamento;
d) por fim a SS registrada pelo cliente terá que ser encerrada manualmente pelo próprio testador após os testes, para evitar que se consuma mais tempo de testes nessa rotina, já que as demais tramitações não possuem qualquer referência clara de que sofrerão impacto de forma diferente do que poderá ser detectado nas tramitações testadas com a SS cadastrada pelo Escritório;
e) é importante considerar que o critério de seleção dos casos se baseará, diante desse cenário, em um filtro pela suíte “3,00-CadastrandoeTramitandoSS”, bem como diretamente pelas opções de Casos de Testes pertinentes a cada tramitação pretendida, já que há uma ordem de precedência necessária para que se garanta o cadastro da SS a ser testada em mais de uma situação distinta;
f) para a falha no cadastro de usuário de clientes, deverá ser testado um caso de teste para o cadastro de usuário de cliente e outro para escritório, aplicando filtro pela suíte “1,00-CadastrandoUsuarios” e pela coluna OPCAO_A_TESTAR, filtrando as opções pertinentes;
g) para a falha no cadastro de notícia deverá existir um caso voltado à gravação da notícia, outro voltado para a edição de uma notícia e outro para a exclusão da

notícia. O primeiro caso visa garantir o funcionamento do que foi corrigido, o segundo e o terceiro visam manter em atividade o que já estava funcionando antes da manutenção realizada. Para isso serão aplicados filtros na matriz de planejamento dos testes focando a suíte "2,00-CadastrandoeExcluindoNoticias", bem como a coluna OPCAO_A_TESTAR, filtrando as opções pertinentes.

Aplicando os critérios acima para a derivação dos casos, dentro da matriz de planejamento dos testes, serão listados os casos de testes visíveis na tabela 8, os quais permitirão, como se pode observar, o término dos testes automatizados em um tempo bastante hábil, se comparado aos mesmos testes realizados manualmente.

Tabela 8 – Casos de testes selecionados na matriz de planejamento dos testes.

Casos de Testes Selecionados	Tempo Manual	Tempo Autom.	Ganho %
3,02-CadastrandoSSInvalida.html	00:00:50	00:00:13	285%
3,03-CadastrandoSSCompletaMeioAcessoEmail.html	00:00:50	00:00:13	285%
3,06-AlterandoSSCompletaMeioAcessoEmail-EmAnalise.html	00:00:50	00:00:13	285%
3,07-AlterandoSSCompletaMeioAcessoEmail-RespondidoEscritorio.html	00:00:50	00:00:13	285%
3,10-QuestionandoSSCompletaMeioAcessoE-mail.html	00:00:50	00:00:13	285%
3,13-AlterandoSSCompletaMeioAcessoEmail-QuestionamentoCliente.html	00:00:50	00:00:13	285%
3,15-ConcluindoSSCompletaMeioAcessoEmail-Satisfeito.html	00:00:50	00:00:13	285%
3,16-CadastrandoSSCompletaPeloCliente-MuitoSatisfeito.html	00:00:50	00:00:13	285%
1,01-CadastroUsuarioSupervisorCliente.html	00:00:36	00:00:07	414%
1,03-CadastrandoUsuarioSupervisorEscritorio.html	00:00:36	00:00:07	414%
2,01-CadstrandoNoticiaValida.html	00:00:28	00:00:03	932%
2,02-EditandoNoticiaValida.html	00:00:28	00:00:03	932%
2,03-ApagandoNoticiaValida.html	00:00:28	00:00:03	932%

Fonte: O próprio autor (2012).

É prudente considerar que para os casos de testes voltados para a publicação de notícias, o ganho acentuado de tempo com os testes automatizados é devido ao fato de que a digitação realizada pela automação é extremamente rápida, pois consiste no ato de copiar e colar dados. Estes casos foram praticados manualmente com a digitação dos textos, com o objetivo de realçar a diferença que o teste automatizado proporciona quando existe um considerável volume de digitação necessária para determinado teste manual.

Avaliando a tabela 7, considerando o tempo total de 9 minutos e 16 segundos para a realização dos testes manuais, percebe-se que a aplicação da matriz de planejamento para a realização dos testes de regressão teve efeito, a tal ponto que mesmo testando manualmente, sobrará tempo para o testador praticar testes exploratórios em outros pontos que considerar seguro efetuar uma avaliação adicional. Contudo analisando o tempo de 2 minutos e 07

segundos que serão consumidos para executar as mesmas tarefas por meio da automação de testes, fica evidente que o testador manual deverá partir diretamente para os testes exploratórios, deixando os testes de regressão a cargo da automação, que terá concluído suas tarefas bem antes do testador manual.

Percebe-se que no cenário hipotético o ganho de tempo proporcionado pela automação de testes chegou à casa dos 338%. Assim, para finalizar a análise dos resultados obtidos e tomando como base o caso hipotético já exposto, supondo que surja um fato novo, nesse caso um erro decorrente de uma das alterações, detectado pela automação, o que exigirá a correção e nova compilação, ainda buscando respeitar o limite de liberar a versão antes das 18h.

Considerando que o especialista de testes tem a ciência de que a automação está atuando de forma muito mais rápida que os mesmos testes manuais, chegará à conclusão de que nesse cenário crítico, em que a pressão poderá induzir o desenvolvedor e o testador a um novo erro refletido em qualquer ponto do software, torna-se seguro optar pela execução da automação de testes regressivos abrangendo toda a base de testes.

Fazendo as contas, a versão veio para testes pela primeira vez ás 17h20min, foi testada pela automação em 02 minutos e 07 segundos, tempo que levou para apontar um erro. Entre a análise do erro, correção e conclusão da compilação passaram-se hipotéticos 13 minutos, assim às 17h45m a versão veio novamente para testes.

Nesse momento o especialista de testes passa a ter a missão de garantir a qualidade de um software que tem que ser liberado em 10 minutos, sendo que está ciente da pressão que a circunstância exerceu sobre os envolvidos.

Tomando por base o tempo total consumido para testes automatizados de toda a versão, que é de 08 minutos e 03 segundos, ao contrário da decisão anterior, nesse momento passa a ser crucial a garantia de que agora o software será entregue com uma maior cobertura das suas funcionalidades, justamente pelos riscos imputados pela pressão do prazo e a falha recente detectada. É adequado considerar que o testador manual poderá seguir atuando em paralelo com testes exploratórios, porém baseado em casos de testes levantados por meio da matriz de planejamento dos testes, focando casos que estejam diretamente ligados às funcionalidades que receberam as últimas correções.

Essa análise fica mais nítida quando se avalia que se os testes forem praticados de forma manual, pela contagem do tempo, em 8 minutos serão cobertos apenas 27 dos 47 casos de testes possíveis, o que representa pouco mais de 50% de cobertura funcional sobre os casos

que compõem a base de testes, frente a uma cobertura de 100% que será proporcionada pelos testes automatizados no mesmo espaço de tempo. Considerando hipoteticamente que a base de testes composta para o trabalho representa 100% da cobertura funcional do software Atendimento, fica clara a capacidade de ampliação da cobertura funcional que os testes automatizados proporcionaram.

Outra constatação que pode ser aplicada diz respeito à capacidade produtiva, comparando o montante que pode ser testado manualmente ou pela automação. Para isso é importante trabalhar com os tempos em segundos, considerando assim um total de 1800 segundos ou 30 minutos para os testes manuais e 480 segundos ou 8 minutos para os testes automatizados, perfazendo assim uma média de 38,29 segundos para os testes manuais e 10,21 segundos para os testes automatizados dos 47 casos que formam a base de testes. Com esses números podemos chegar ao seguinte cálculo:

a) em 30 minutos de testes, supondo que fosse mantida a média de tempo da base atual, os testes automatizados conseguiriam cobrir 176 casos de testes, o que se obtém pela divisão do total de 1800 segundos que seriam consumidos em testes manuais pela média de 10,21 segundos dos testes automatizados;
b) por outro lado, para conseguir cobrir os mesmos 176 casos de testes que seriam alcançados pela automação em 30 minutos, porém agora praticando testes manuais, seriam necessários 6739 segundos, ou seja, 1 hora e 52 minutos.

É válido considerar que os exemplos críticos expostos serviram apenas para enfatizar os resultados que se esperava obter, pois indiferente às circunstâncias propostas, o fato relevante é que de forma prática foi constatado que a automação de testes, em um mesmo espaço de tempo, permite uma maior cobertura funcional, ampliando a quantidade de casos de testes que possam ser exercitados, exatamente como foi observado por Fewster e Graham (1999, tradução nossa) no diagrama exposto na Figura 8 do subcapítulo 4.1.

O subcapítulo 4.1 trata das dificuldades enfrentadas na tentativa de se implantar um processo de automação de testes de software. Ainda segundo Fewster e Graham (1999, tradução nossa), uma das maiores limitações nesse sentido é a seleção de casos de testes que sejam econômicos, eficientes, reutilizáveis e evolutivos, sendo que somente casos de testes que contemplem tais características apresentarão como resultado o crescimento da sua capacidade de cobertura funcional, pelo aumento de casos de testes executados em um dado espaço de tempo, o que pôde ser observado por meio dos experimentos realizados.

CONCLUSÃO

A crescente dependência das organizações atuais por soluções de software remetem a uma responsabilidade maior por parte dos desenvolvedores no que tange a qualidade dos produtos desenvolvidos. Acredita-se que a qualidade do produto está diretamente relacionada à qualidade do processo ao qual foi submetido, tornando essencial a busca pela melhoria contínua dos processos de desenvolvimento.

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

Também foi constatado que entre as técnicas de testes, a funcional ou caixa-preta se mostra relevante pela capacidade de validar os requisitos funcionais. Dentre as fases de testes, mereceu destaque a de regressão, justamente por assegurar a aderência do software aos requisitos funcionais originalmente projetados, indiferente ao tempo decorrido desde o desenvolvimento original.

Em outra linha de estudo, o trabalho denotou o fato de que softwares que sofrem constantes manutenções são mais propensos a apresentar erros, tornando-os mais carentes de testes a cada atualização, por menor que esta seja. Tal informação constata que a automação de testes funcionais regressivos é uma alternativa viável e aplicável para tais softwares.

Tomando isso como base, o trabalho foi elaborado visando demonstrar a capacidade de ampliação da cobertura funcional que a automação de testes funcionais regressivos poderia proporcionar.

Usando o critério de derivação baseado no conhecimento e experiência do especialista, a base de testes e uma matriz para o planejamento dos testes foram compostas, esta última formada por colunas correspondentes aos requisitos funcionais cobertos pelos casos de testes, visando futuras seleções de casos de testes compatíveis com os requisitos funcionais modificados em uma hipotética manutenção.

Após essa etapa, os casos de testes foram automatizados com a geração dos *scripts*, utilizando para isso a ferramenta de automação de testes funcionais *web* denominada Selenium IDE, sendo que o software Domínio Atendimento foi a solução *web* a ser testada.

Empregando a matriz de planejamento, experimentos foram realizados com a execução dos casos de testes e anotação dos tempos praticados, de forma automática e manual.

Por fim, com a sugestão de situações hipotéticas, constatou-se um ganho de 281% no uso dos testes automatizados frente à prática dos testes manuais, demonstrando a capacidade da automação de testes em ampliar a cobertura funcional, quando comparada aos testes manuais em um mesmo espaço de tempo. Isso também levou à compreensão de que a metodologia aplicada no trabalho gerou casos de testes econômicos, eficientes e reutilizáveis, o que para softwares de manutenções frequentes representa maior produtividade e segurança, proporcionando inclusive uma redução no número de novas correções e ampliando a qualidade do produto, permitindo concluir que o trabalho atingiu seu objetivo geral.

Como contribuição científica, é importante considerar que o trabalho permitirá estudos futuros voltados ao uso de critérios formais na derivação dos casos de testes ou de uma matriz de rastreabilidade dos testes, os quais, combinados com a automação de testes regressivos, tendem a apresentar ganhos adicionais à cobertura funcional.

REFERÊNCIAS

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

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

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

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

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

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

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

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

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

CRESPO, Adalberto Nobiato et al. **Uma Metodologia Para Testes de Software no Contexto de Melhoria do Processo.** 15 f. Artigo-Centro de Pesquisas Renato Archer (CENPRA). Campinas, 2012. Disponível em: <bibliotecadigital.sbc.org.br/download.php?paper=254>.
Acesso em: 27/06/2012.

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

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

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

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

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

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

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

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

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

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

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

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

MYERS, Glenford J. ***The Art of Software Testing*.** New Jersey: John Wiley & Sons, Inc., 2004.

TENÓRIO JUNIOR, Nelson Nunes. **Modelo–E10: Um modelo para estimativas de esforço em manutenção de software.** 134 f. Tese (Doutorado em Ciência da Computação)-Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre, 2009.

PETERS, James F.; PEDRYCZ, Witold. ***Software engineering : an engineering approach*.** USA: John Wiley & Sons, Inc., 2000.

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

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

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software: 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.

SEI, Software Engineering Institute. **CMMI *for Development, Version* 1.3.** USA: 2010. Disponível em: <http://www.sei.cmu.edu/reports/10tr033.pdf>. Acesso em: 17/10/2012.

SELENIUM. **Selenium *Web Application Testing System*.** Disponível em: < http://seleniumhq.org/>. Acesso em: 13/09/2012.

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

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

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

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

APÊNDICE A – ARTIGO

Testes Automatizados para Ampliar a Cobertura Funcional na Manutenção de Software

Cleverson Reinert[1]

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

Criciúma – SC – Brasil

cleverson78@gmail.com

Abstract. *The quality is a necessity sought by software engineering professionals, who try to evolve processes and products, which are used in critical routines of daily life. It is also known that frequent maintenance expose the software to a greater number of errors. This validates studies on the use of regressive functional testing automated aimed at increasing the coverage of functional software with constant maintenance. The use of resources that enhance the efficiency of this coverage is also shown important, including comparing the results of automated testing with manual testing.*

Resumo. *A qualidade é uma necessidade buscada por profissionais da engenharia de software, que tentam evoluir processos e produtos, os quais são empregados em rotinas críticas do cotidiano. Também é sabido que frequentes manutenções expõem os softwares a um número maior de erros. Isso valida estudos acerca do uso da automação de testes funcionais regressivos que visem ampliar a cobertura funcional de softwares com constantes manutenções. O emprego de recursos que ampliam a eficiência dessa cobertura também se mostra importante, inclusive comparando os resultados de testes automatizados com testes manuais.*

1. Introdução

Estudos da qualidade de software são importantes, pois tais produtos estão presentes em nosso cotidiano, sendo a qualidade afetada por fatores como a heterogeneidade de ambientes, sistemas legados e os prazos reduzidos (SOMMERVILLE, 2003), tornando pertinente considerar que Segundo Pressman (2006), quanto mais frequentes são as manutenções do software, maiores são as chances de ocorrer erros.

O uso de testes funcionais visa confirmar a aderência dos produtos aos requisitos funcionais solicitados (DELAMARO; MALDONADO; JINO, 2007). Tais testes possuem a etapa de regressão, garantindo que requisitos funcionais sigam operando após uma manutenção (PEZZÈ; YOUNG, 2008).

É pertinente considerar que testes funcionais tendem a onerar o processo produtivo se aplicados de forma manual e exaustiva (MYERS, 2004, tradução nossa).

Assim, reduzir o tempo de desenvolvimento e os custos produtivos se mostra importante. Para isso pode ser aplicada a ampliação da cobertura funcional em um espaço de tempo, empregando testes automatizados. Tal recurso também tem seus contratempos, pois é um investimento de longo prazo que exige reestruturar equipes, contratar e às vezes adquirir softwares e equipamentos (RIOS; MOREIRA, 2006).

Tendo em vista as necessidades da qualidade do software e as limitações para implantar uma automação, este meta-artigo apresenta estudos da automação de testes funcionais regressivos que ampliem a cobertura funcional, reduzam os defeitos, o tempo de testes e os custos produtivos, proporcionando um aumento da qualidade.

2. Justificativa

O uso de softwares se tornou tão constante que só percebemos sua existência diante de uma falha, como por exemplo, ao usar caixa automático que emita a frase "Falha geral no sistema, utilize outro caixa.".

Este exemplo demonstra o quão importante é o estudo da engenharia de software, a qual emprega princípios científicos, métodos, modelos, padrões e teorias que possibilitam gerenciar, planejar, modelar, desenhar, implementar, mensurar, analisar, manter e evoluir sistemas de software, tudo isso com a produção econômica de softwares qualificados (PETERS; PEDRYCZ, 2000, tradução nossa).

Segundo apontam Pezzè e Young (2008), um software e um carro de corrida são produtos únicos, exigindo a avaliação da qualidade para cada novo exemplar. Ainda assim as falhas de software estão presentes desde o início do projeto e devido a isso a fase de testes compreende diversas etapas de validação (DELAMARO; MALDONADO; JINO, 2007). Para isso existem, entre outras técnicas, os testes caixa-branca e caixa-preta, este último também chamado de funcional (DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006; SOMMERVILLE, 2003).

A tarefa de testes permite combinar várias técnicas visando a qualidade, contudo a de testes funcionais é um dos focos deste meta-artigo, justamente porque segundo Sommerville (2003) permite testes bastante abrangentes sobre os requisitos funcionais.

A abordagem de teste sistemático com uso de critérios, de acordo com Pezzé e Young (2008), é a alternativa ideal para derivar casos de testes, pois reduz os esforços e possibilita maior abrangência na cobertura funcional. Mesmo assim, o teste é uma etapa que pode consumir 50% do tempo total de entrega do produto, exigindo o uso de aplicativos para automação de teste (SOMMERVILLE, 2003), o qual reduz incomensuravelmente o tempo de teste e amplia a qualidade (PEZZÈ; YOUNG, 2008).

Frente aos fatores expostos e considerando-se a crescente busca da qualidade sem prejuízo a prazos de projetos, este meta-artigo mostra-se relevante, já que os testes automatizados mostram-se viáveis para ampliar a cobertura funcional e elevar a qualidade desejada, reduzindo o tempo final do projeto de software.

3. Automação de Testes Regressivos na Ampliação da Cobertura Funcional de Softwares com Manutenções Frequentes

A tentativa de garantir a qualidade no produto de software invariavelmente passa pela necessidade de submetê-lo a testes (MYERS, 2004, tradução nossa), inclusive funcionais (PRESSMAN, 2006). Da mesma forma também é importante relembrar que manutenções constantes deixam o software propenso a apresentar defeitos (SOMMERVILLE, 2003) e que o custo para aplicar testes de regressão de forma exaustiva é alto (MYERS, 2004, tradução nossa), tornando a tarefa impraticável de maneira manual.

Os subcapítulos seguintes explorarão em detalhes as etapas de criação deste artigo, expondo as tarefas realizadas e por fim, os resultados obtidos.

3.1. Estudo Sobre as Técnicas, Ferramentas e Modelos de Qualidade da Engenharia de Software

Tomando como base os livros de diversos autores foram estudados temas inerentes à engenharia de software, abordando modelos, ferramentas e técnicas para o gerenciamento de projetos, enfocando softwares com manutenções frequentes.

Isso permitiu observar que quanto mais intervenções no software, crescem os erros (PRESSMAN, 2006), ao passo que segundo Ghezzi, Jazayeri e Mandrioli (1991, tradução nossa), cerca de 60% do esforço de produção envolve manutenções, o que confirma que entre os fatores da qualidade produtiva está a manutenabilidade, a qual é composta principalmente pela facilidade dos reparos e a praticidade da evolução do software (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa).

As pesquisas com normatizações e modelos de qualidade, como por exemplo, o CMMI (SEI, 2010, tradução nossa) na versão 1.3, MPS.br (SOFTEX, 2011) e NBR ISO/IEC 9126 (ABNT, 2003), deram um enfoque sobre a manutenabilidade e a tarefa de testes de software, validando a relevância do teste na qualidade, denotando a necessidade de se testar da forma mais eficaz e produtiva possível.

Aliando o conhecimento adquirido acerca da engenharia de software, qualidade de software e teste de software, ao o fato de que, segundo Sommerville (2003), a tarefa de testes pode consumir cerca de 50% do tempo de produção para uma versão, tornou-se importante um estudo sobre a tarefa de teste caixa-preta, suas principais técnicas, métodos e fases, o que será abordado no próximo subcapítulo.

3.2. Estudo Sobre a Tarefa de Teste Caixa-preta

Embasado em publicações pesquisadas, estudos mostraram os conceitos do teste de software, técnicas e fases, bem como suas aplicações na busca da qualidade.

A técnica de teste caixa-preta ou funcional destacou-se por atuar com enfoque na validação dos requisitos funcionais do produto (PRESSMAN, 2006).

O estudo apresentou a divisão de fases do teste funcional, sendo que a de regressão foi a opção escolhida para os experimentos do presente artigo, isso porque segundo Pezzè e Young (2008), seu diferencial está no impacto em softwares de manutenção frequente, pois assegura que casos de testes validados manualmente em versões anteriores, sigam válidos na versão em teste.

O estudo mostrou que o teste caixa-preta de regressão aumenta a qualidade do software ao manter a aderência do mesmo aos requisitos funcionais originais por meio da ampliação da cobertura funcional, um dos principais objetivos do presente artigo. Assim o estudo dos critérios de derivação se tornou tema do próximo subcapítulo.

3.3. Estudo e Seleção do Critério de Derivação dos Casos de Testes

Os estudos apresentaram o conceito, possíveis aplicações e até as limitações dos critérios formais e não formais de derivação dos casos de teste caixa-preta.

A pesquisa também mostrou a importância dos testes funcionais que minimizem os prejuízos quanto a prazos e qualidade, ocasionados por testes além ou aquém do necessário (RIOS; MOREIRA, 2006) e que testes funcionais exaustivos são impraticáveis (MYERS, 2004, tradução nossa).

O uso de critérios formais de derivação, baseados em particionamento de equivalências, análise do valor limite, entre outros, podem ser empregados de forma complementar entre si. Também foi possível constatar que adotar algum critério, quer seja ele formal ou não formal, é fator crucial para maior abrangência funcional, sendo aceito, segundo Rios e Moreira (2006), o uso do critério não formal baseado no conhecimento e experiência do especialista.

Após as pesquisas o critério não formal baseado no conhecimento e experiência do especialista foi escolhido para a realização dos experimentos, o que ocorrerá nas fases de composição da base de testes e da matriz de planejamento dos testes. Já o subcapítulo seguinte trata dos estudos praticados para escolha do software de automação.

3.4. Estudo e Seleção da Ferramenta de Automação dos Testes Funcionais

Os estudos demonstraram as características das ferramentas Testcomplete, Autoit e Selenium IDE, sendo que todas possuíam o suporte à gravação e execução de *scripts* de automação.

A ferramenta Testcomplete se mostrou como a única opção comercializada, atuando em testes com ambientes *desktop* e *web.* Entre as outras duas ferramentas, ambas *open source*, as pesquisas demonstraram que tanto o software Autoit (AUTOIT, 2012, tradução nossa) quanto o Selenium IDE (SELENIUM, 2012, tradução nossa) possuem as funções necessárias para uma prática adequada e segura de testes funcionais, contudo o primeiro foca ambientes *desktop* e o segundo ambientes *web*.

Para o intento didático deste artigo, a aquisição do software Testcomplete foi dispensada, sendo que a opção escolhida foi o Selenium IDE, tanto pelo interesse científico no software, como pela praticidade a segurança que a ferramenta apresenta, como por exemplo, durante a fase de estudos e escolha do software a ser testado para realização do artigo, tópico este que será abordado no próximo subcapítulo.

3.5. Estudo e Seleção do Software *web* a Testar

Os estudos iniciais da ferramenta Selenium IDE permitiram observar incompatibilidades e falhas de *scripts*, como por exemplo, com o texto “Element link=modelo not found” para o site “ead.unesc.com.br” na sessão de download dos materiais de aula, mensagens estas que foram observadas em outros softwares testados.

Os estudos e testes realizados não permitiram resolver as falhas, contudo é válido afirmar que o Selenium IDE possua recursos que as contornem, devido à sua robustez e diversidade de recursos. Contudo a importância quanto ao andamento do estudo levou à necessidade confirmar alguma plataforma de desenvolvimento que não apresentaria tais críticas, identificando a linguagem Java como tal. Após a constatação o software avaliado e escolhido foi o Domínio Atendimento, da empresa Domínio Sistemas Ltda, com sede em Criciúma/SC, o qual é desenvolvido com a linguagem Java e baseado em interface gráfica.

O Domínio Atendimento foi pioneiro em seu gênero no Brasil dando suporte ao relacionamento entre escritórios contábeis e seus clientes. É considerado um dos módulos do sistema Domínio Contábil Plus, também bastante difundido nacionalmente.

O principal recurso do Atendimento está na opção de registro das Solicitações de Serviços (SS) por clientes ou escritório, o que permite solicitar, por exemplo, emissão de contratos, geração de declarações, cálculos de férias e rescisões, entre outros. Após registrar uma SS, os usuários do escritório que operam o Domínio Contábil Plus, usado em *Desktop*, recebem *online* a notificação da SS, realizam tarefas requisitadas e do ambiente *Desktop* enviam a solução. Ao concluir a SS o cliente vota quanto à satisfação obtida, escolhendo entre Satisfeito, Muito Satisfeito, Insatisfeito, Muito Insatisfeito ou Desejo Não Opinar. Tais dados geram um gráfico permitindo ao escritório avaliar o desempenho da equipe quanto aos atendimentos.

A geração dos casos da base de testes, tema do subcapítulo seguinte, buscou combinar apenas algumas das funcionalidades visando manter o objetivo didático.

3.6. Composição da Base de Testes

A etapa ocorreu com a documentação dos casos de testes que cubram funcionalidades mais críticas, levando em conta a frequência de manutenções recebidas e o uso constante da rotina, as quais precisassem de validações constantes.

Assim, foram delimitadas as funcionalidades a testar empregando o critério não formal com base no conhecimento e experiência do especialista, inclusive aplicando conhecimentos sobre a qualidade de software, testes funcionais regressivos.

A tabela 4 demonstra as suítes de testes geradas, funcionalidades contempladas e totais de casos.

Analisando a tabela 4, merece destaque a suíte de testes "3,0-CadastrandoeTramitandoSS", a qual contempla o maior número de casos de testes, pois comporta opções ligadas às SS, sendo um dos requisitos funcionais mais usados.

A escolha da cobertura de funcionalidades como o acesso negado (ou permitido), cadastro de usuários e cadastro de notícias levou em conta o fato de que um erro em qualquer uma destas seria crítico. Aplicando critério igual constatou-se que para as SS os requisitos funcionais não precisam ter todas as variações cobertas.

Com os 47 casos de testes estruturados na base de testes, tornou-se necessário compor a matriz de planejamento dos testes, o tema do subcapítulo seguinte.

Tabela 4 – Base de testes sumarizada.

Suíte De Teste	Funcionalidades Abrangidas	Número de Casos de Testes
0,0-TestandoAcessos	Acesso negado e acesso permitido (com usuários Supervisor de Escritório e Supervisor de Cliente)	6
1,0-CadastrandoUsuarios	Cadastro válido, tentativa de cadastro inválido (usuários supervisor de Cliente e supervisor de Escritório)	4
2,0-CadastrandoeExcluindoNoticia	Cadastro válido (notícia para funcionários e notícia para clientes), tentativa de cadastro inválido, visualização pelo cliente e exclusão de notícia.	7
3,0-CadastrandoeTramitandoSS	Cadastro pelo Escritório, trâmites Em Análise, Em Análise Escritório, Aguardando Resposta, Respondido (Cliente), tentativa de visualizar trâmite bloqueado (Cliente), Concluído (Escritório), cadastro pelo Cliente, Concluído pelo cliente (com votações de Satisfeito, Muito Satisfeito, Insatisfeito, Muito Insatisfeito e Não Desejo Opinar).	27
999,0-CasosdeTestesUsoGeral	Fechamento de sessão, acesso com usuários Supervisores (Cliente e Escritório)	3

3.7. Composição da Matriz de Planejamento dos Testes

A aplicação do critério de derivação baseado no conhecimento experiência do especialista permitiu elaborar uma matriz de planejamento dos testes que contemplasse colunas com diversos requisitos funcionais.

A matriz possui a finalidade de permitir filtros dinâmicos de requisitos que passem por ma manutenção, permitindo assim a seleção de casos que enfoquem apenas as funcionalidades afetadas, permitindo delimitar o raio de ação dos testes funcionais regressivos, sejam eles manuais ou automatizados.

Para algumas das colunas foram determinadas informações de identificação dos casos de testes automatizados, como por exemplo, as colunas Suite e Casos_deTestes.

Os requisitos listados em colunas receberam os nomes visíveis de forma esquemática na Figura 2.

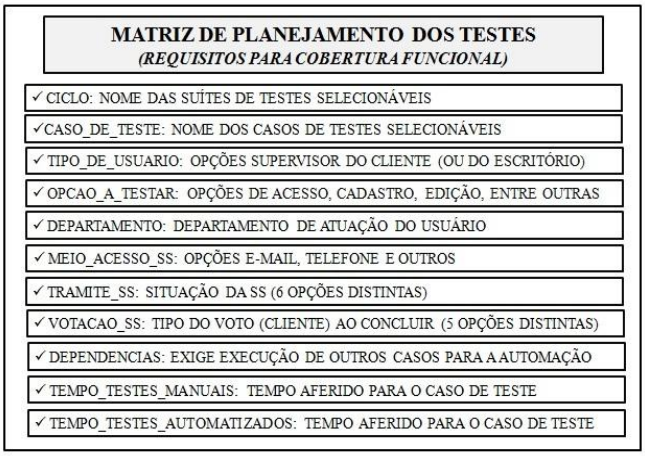


Figura 2. Composição da matriz de planejamento dos testes.

Merecem destaque na figura 2 as colunas Tempo_Testes_Manuais e Tempo_Testes_Automatizados, pois são os tempos aferidos durante as etapas de composição da base de testes e de gravação da automação de testes, os quais servem para o especialista avaliar o tempo que determinado teste consumirá.

Os tempos coleados tiveram papel importante para os estudos do trabalho, o que será mais bem observado na análise dos resultados obtidos. O subcapítulo seguinte abordará a geração dos *scripts* de casos de testes, bem como a execução da automação.

3.8. Execução dos Testes Manuais com Gravação da Automação, Criação dos *Scripts* e Execução da Automação de Testes

Com uso da ferramenta Selenium IDE, foram realizados os testes manuais no software Domínio Atendimento a partir do navegador *web* Mozilla Firefox, com a gravação acionada pelo menu Ações e a opção Gravar do Selenium IDE, selecionando cada caso com base na matriz de planejamento, respeitando-se a ordem de precedência.

A primeira execução consumiu 35% a mais de tempo devido às tarefas de gravação dos *scripts*, reforçando que só da segunda execução de testes automatizados em diante é que surgem ganhos (FEWSTER; GRAHAM, 1999, tradução nossa).

Após gravar os *scripts* em HTML, a execução das automações ocorreu com a abertura das suítes de teste, acionando o menu Ação e a opção Play do Selenium IDE.

A execução da automação permitiu observar diversas características que dão destaque para o uso da automação de testes funcionais regressivos na ampliação da cobertura funcional, o que será o tema central do subcapítulo a seguir.

4. Resultados Obtidos:

Com base em experiências anteriores e na bibliografia, havia a expectativa de que resultados positivos seriam atingidos. As primeiras execuções de testes com tomadas de tempo apontaram essa tendência, com os testes manuais aplicados em toda a base de testes consumindo 30 minutos e 41 segundos (1841 segundos), sem as tarefas de gravação de *scripts* envolvidas, enquanto que a automação consumiu 8 minutos e 03 segundos (483 segundos).

A tabela 2 apresenta as diferenças detectadas entre as suítes de testes que compõem a base de testes.

Tabela 6 – Testes manuais X automatizados.

Suítes de Testes	Tempo dos Testes Manuais	Tempo dos Testes Automatizados	Ganho % com Testes Automatizados
0,0-TestandoAcessos	00:01:42	00:00:30	240%
1,0-CadastrandoUsuarios	00:02:33	00:00:36	325%
2,0-CadastrandoeExcluindoNoticia	00:03:45	00:00:41	449%
3,0-CadastrandoeTramitandoSS	00:22:41	00:06:16	262%

Os números mostram que com o ganho da automação perto de 281%, quando o testador manual está próximo a 1/3 da execução a automação já se encerrou.

Um experimento hipotético pode considerar que o software Atendimento será alterado no cadastro de SS registrada pelo Escritório, invertendo os campos Usuário e Departamento. Também será corrigido o cadastro de usuários de clientes, eliminando um erro ao gravar e o cadastro de notícias, retirando uma falha de gravação do texto.

Tal versão veio para testes às 17h20min, tendo que ser liberada até às 17h50min. Para fins didáticos, supõem-se que a base de testes representa 100% de cobertura funcional, assim sabe-se que testar manualmente será impossível, pois levaria 30 minutos.

As correções exigem o levantamento dos requisitos a exercitar regressivamente, usando a matriz de planejamento e o conhecimento e experiência do especialista.

A tabela 3 demonstra os casos de testes selecionados na matriz de planejamento, sendo que avaliando o tempo de 9 minutos e 16 segundos para os testes manuais nota-se que a matriz de planejamento teve efeito para os testes de regressão manuais, contudo o tempo de 2 minutos e 07 da automação mostra que o testador manual deve fazer testes exploratórios, deixando a regressão a cargo da automação.

No cenário hipotético, o ganho de tempo proporcionado pela automação de testes chegou à casa dos 338%, proporcionando ampliação da cobertura funcional e permitindo melhor aproveitamento dos testes manuais visando maior qualidade.

Em outra análise, focando as médias de 10,21 segundos da automação versus 38,29 segundos manuais para os 47 casos da base de testes, pode-se concluir que:

a) em 30 minutos ou 1800 segundos de testes, mantendo-se a média da base atual, os testes automatizados cobririam 176 casos de testes;
b) por outro lado, para cobrir os mesmos 176 casos, os testes manuais consumiriam 6739 segundos, ou seja, 1 hora e 52 minutos.

Os experimentos hipotéticos acima permitiram avaliar que a automação de testes, em um mesmo espaço de tempo, permite uma maior cobertura funcional, ampliando a quantidade de casos de testes que possam ser exercitados, exatamente como foi observado por Fewster e Graham (1999, tradução nossa).

Tabela 8 – Casos de testes selecionados na matriz de planejamento dos testes.

Casos de Testes Selecionados	Tempo Manual	Tempo Autom.	Ganho %
3,02-CadastrandoSSInvalida.html	00:00:50	00:00:13	285%
3,03-CadastrandoSSCompletaMeioAcessoEmail.html	00:00:50	00:00:13	285%
3,06-AlterandoSSCompletaMeioAcessoEmail-EmAnalise.html	00:00:50	00:00:13	285%
3,07-AlterandoSSCompletaMeioAcessoEmail-RespondidoEscritorio.html	00:00:50	00:00:13	285%
3,10-QuestionandoSSCompletaMeioAcessoE-mail.html	00:00:50	00:00:13	285%
3,13-AlterandoSSCompletaMeioAcessoEmail-QuestionamentoCliente.html	00:00:50	00:00:13	285%
3,15-ConcluindoSSCompletaMeioAcessoEmail-Satisfeito.html	00:00:50	00:00:13	285%
3,16-CadastrandoSSCompletaPeloCliente-MuitoSatisfeito.html	00:00:50	00:00:13	285%
1,01-CadastroUsuarioSupervisorCliente.html	00:00:36	00:00:07	414%
1,03-CadastrandoUsuarioSupervisorEscritorio.html	00:00:36	00:00:07	414%
2,01-CadstrandoNoticiaValida.html	00:00:28	00:00:03	932%
2,02-EditandoNoticiaValida.html	00:00:28	00:00:03	932%
2,03-ApagandoNoticiaValida.html	00:00:28	00:00:03	932%

5. Conclusão

A dependência das organizações por soluções de software gera maior responsabilidade por parte dos desenvolvedores, tornando essencial a busca pela melhoria contínua dos processos de desenvolvimento.

O artigo permitiu observar as contribuições existentes nas metodologias, modelos e normatizações, sendo que a validação da qualidade emprega o uso de testes.

O emprego de testes funcionais se mostrou relevante por validar funcionalidades, mas estes apresentam limitados se aplicados de forma manual e exaustiva, principalmente em softwares de constantes manutenções. Nesse contexto o uso de testes funcionais regressivos mostrou-se adequado para manter o produto condizente com seus requisitos diante de possíveis manutenções. Como a aplicação de testes funcionais regressivos e exaustivos também é impraticável, a adoção de testes funcionais regressivos automatizados mostrou-se a alternativa adequada.

Aplicando uma matriz de planejamento, com uso de critério baseado no conhecimento e experiência do especialista de testes, uma base de testes foi estruturada e submetida à execução de testes manuais e automatizados, permitindo observar, com base nos experimentos hipotéticos, que os ganhos da automação chegaram à casa dos 281%, proporcionando a ampliação da cobertura funcional e permitindo o alcance dos objetivos centrais do presente meta-artigo.

6. Referências

ABNT, Associação Brasileira de Normas Técnicas. **NBR ISO/IEC 9126-1 -**

Engenharia de software - Qualidade de produto - Parte 1: Modelo de qualidade.

Rio de Janeiro: ABNT, 2003.

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

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

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

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

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

MYERS, Glenford J. ***The Art of Software Testing*****.** New Jersey: John Wiley & Sons, Inc., 2004.

PETERS, James F.; PEDRYCZ, Witold. ***Software engineering : an engineering approach*****.** USA: John Wiley & Sons, Inc., 2000.

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

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

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software: Segunda edição revisada e ampliada.** Brasil: Alta Books, 2006.

SEI, Software Engineering Institute. **CMMI** ***for Development, Version*** **1.3.** USA: 2010. Disponível em: <http://www.sei.cmu.edu/reports/10tr033.pdf>. Acesso em: 17/10/2012.

SELENIUM. **Selenium** ***Web Application Testing System*****.** Disponível em: < http://seleniumhq.org/>. Acesso em: 13/09/2012.

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