UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

CAMILLA DAMIANI

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

CRICIÚMA

2012

CAMILLA DAMIANI

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

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

CAMILLA DAMIANI

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

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, 29 de Novembro de 2012.

BANCA EXAMINADORA

Prof. MSc. Gustavo Bisognin – UNESC – Orientador

Prof. Esp. Gilberto Vieira da Silva - UNESC

Prof. MSc. Paracelso de Oliveira Caldas - UNESC

Dedico este trabalho a todos que me ajudaram a concluir esta etapa da minha vida, em especial a minha família, que com amor e paciência sempre esteve disposta a me ajudar. A Deus por sempre guiar meu caminho e me conduzir pela Luz da sua Paz.

.

AGRADECIMENTOS

Quero agradecer a Deus por iluminar todos os dias o meu caminho, guiar os meus passos e me dar forças durante esta caminhada.

Aos meus pais que nunca mediram esforços para me dar a melhor educação, e que sempre acreditaram nos meus objetivos. Principalmente a minha mãe que me acompanha sempre, a quem me espelho para ser uma pessoa melhor, e que é meu anjo da guarda.

Aos meus irmãos Luidj e Muryell,a quem me orgulho muito e que serão sempre meus exemplos.

Ao meu namorado, Diogo, pelo carinho e compreensão demonstrados nessa longa caminhada.

Agradeço, em especial, o meu orientador, Prof. MSc Gustavo Bisognin, o qual tenho grande admiração e profundo respeito, e que desde o início me deu força e me auxiliou muito na criação e elaboração deste trabalho.

A empresa Betha Sistemas pelo apoio fornecido. Aos meus colegas de trabalho pelo incentivo e pelos ensinamentos obtidos até hoje.

Por fim, a todos os professores e colegas do curso de Ciência da Computação.

Agradeço a todos que diretamente e indiretamente contribuíram para a conclusão deste trabalho tão importante.

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

Robert Dunn

RESUMO

Com o crescimento da demanda por sistemas computacionais, observa-se que a exigência por software que apresentam um grau comprovado de qualidade, é cadavez maior. A atividade de teste tem papel fundamental em certificar essa qualidade e identificar defeitos no software precocemente, diminuindo assim o trabalho dos desenvolvedores em corrigi-los posteriormente quando o software já está sendo distribuído aos clientes. Neste contexto, o presente trabalho propõe a construção de uma metodologia que se baseia na automação dos testes para a garantia da cobertura funcional da qualidade dos softwares.

A metodologia foi formada principalmente pela criação do ciclo de vida do teste, juntamente com o planejamento da matriz de cobertura e a criação dos testes automatizados. Foram criadas baterias de testes utilizando casos de testes especificados com o conhecimento do especialista. Para a automatização dos testes foi utilizada a técnica de *capture-replay*na ferramenta *TestComplete*, que permite a criação e execução de testes de forma rápida e vantajosa.

Palavras-chave: Qualidade de Software. Teste de Software. Automação de Teste.

ABSTRACT

With the growing demand for computer systems, it is observed that the demand for software that have a proven level of quality is increasing. The testing activity plays a key role in ensuring that quality and identify defects early in the software, thus reducing the work of the developers fix them later when the software is now shipping to customers. In this context, this paper proposes the construction of a methodology that relies on automation of tests to ensure the functional coverage of software quality.

The methodology was formed mainly by the creation life-cycle test, along with the array coverage planning and creation of automated testing. Batteries of tests were created using test cases specified with the specialist knowledge. To test automation technique was used to capture-replay tool in TestComplete, which enables the creation and execution of tests quickly and advantageous.

Keywords: Software Quality. Software Testing.Test Automation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BVA	Boundary Value Analysis
CCMI	Capability Maturity Model Integration
ISO/IEC	International Organization for Standardization and the International Electrotechnical Commission
MPS.BR	Melhoria de Processos do Software Brasileiro
SEI	Software Engineering Institute
CMMI	Capability Maturity Model Integration
TRM	Traceability Rastreability Matrix

SUMÁRIO

1 INTRODUÇÃO

O mercado atual apresenta uma grande quantidade de sistemas computacionais com baixa qualidade. A realidade das organizações desenvolvedoras de sistemas computacionais é bastante crítica, apresentando projetos com escopo complexo e prazos curtos, o que fomenta o desenvolvimento de software sob pressão e aumenta consideravelmente a probabilidade de inserção de defeitos no processo de desenvolvimento.

Outro fator bastante crítico referente à qualidade dos sistemas desenvolvidos atualmente, é a falta de cobertura de testes, fazendo com que a responsabilidade sob a qualidade da solução fique totalmente com o desenvolvedor, que muitas vezes, abdica da qualidade em detrimento do prazo de entrega.

Com o crescimento da demanda por sistemas computacionais, observa-se que a exigência por software que apresentam um grau comprovado de qualidade, é cada vez maior. Neste contexto, entra a Engenharia de Software com os modelos de qualidade que possuem o objetivo de apoiar o processo de desenvolvimento a fim de garantir a qualidade, desde a fase de concepção até a liberação para a produção.

Segundo Prikladnicki (2006) o desenvolvimento de sistemas de software parece estar relacionado a uma técnica ortodoxia, sendo visto apenas como um processo totalmente técnico a ser executado especificamente por especialistas. O esforço necessário para a produção de sistemas deste tipo apresenta problemas e desafios de complexidade muito além da técnica. Ou seja, necessitam de conhecimentos avançados provenientes de aspectos multidisciplinares, oriundos de outras áreas do conhecimento.

Uma das formas de obter a complementação perfeita entre o conhecimento multidisciplinar e as técnicas de engenharia de software existentes no mercado, é a utilização da experiência do analista de negócio aplicada à composição da massa de testes no processo de derivação funcional. Este aspecto torna-se de fundamental importância, uma vez que, os custos associados à qualidade são bastante elevados, fazendo com que a utilização de testes automatizados se torne uma realidade cada vez mais constante nas organizações desenvolvedoras de software.

O presente trabalho propõe uma metodologia baseada na experiência do analista de negócios para a realização da automação do processo de teste de software como estratégia para garantir a qualidade dos sistemas desenvolvidos.

1.1OBJETIVO GERAL

Desenvolver uma metodologia, baseada na automação do processo de testes de software para garantir a cobertura funcional durante o processo de validação.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste Trabalho de Conclusão de Curso (TCC) são:

a) estudar da área de Qualidade de Software;
b) estudar de ferramentas de automação de testes;
c) compreender o processo de teste de software;
d) definir a ferramenta de automação de testes a ser abordada;
e) desenvolver uma metodologia para a garantia da cobertura funcional de testes de software;
f) desenvolver um projeto piloto para a validação da metodologia abordada.

1.3 JUSTIFICATIVA

Segundo Avizieniset al (2004), uma falha é um evento que ocorre quando o serviço fornecido por um sistema não corresponde à função que o sistema deveria desempenhar.

Um serviço proposto por um sistema computacional é determinado falho quando não corresponde à especificação funcional do sistema ou quando a especificação não define corretamente a função do sistema. Tal situação é denominada erro. (AVIZIENIS ET AL,2004).

As falhas que ocorrem durante a utilização de um sistema, geralmente são causadas pela má implementação da funcionalidade ou por falha na

especificação da mesma, podendo ainda ser associado à confiabilidade. Sistemas instáveis causam prejuízos como perda de dados, atrasos em atividades importantes, re-trabalho, dentre outros.

Segundo Yourdon (1990), não existem sistemas totalmente seguros ou livres de falhas. Devido à grande quantidade de funcionalidades pouco utilizadas em aplicações de software, alguns erros nunca chegam a serem descobertos, fazendo com que o software permaneça eternamente com defeito.

A cobertura funcional automática aumenta a quantidade de validações de funcionalidades e dependências, fazendo com que uma quantidade maior de defeitos seja identificada na fase de desenvolvimento, diminuindo custo e aumentando a satisfação dos usuários do sistema desenvolvido.

As ferramentas para construção de testes automáticos funcionais permitem a realização dos mesmos de uma forma não assistida, aumentando a capacidade de cobertura em um curto período de tempo. Atualmente existem diversas ferramentas no mercado que são aplicadas na automação de testes funcionais de software.

Algumas delas apresentam um funcionamento denominado capture-replay ou gravação-execução. Estas ferramentas permitem a criação, execução e a verificação dos resultados de testes automáticos para sistemas com interface gráfica *standar*. Outras técnicas utilizadas podem ser citadas, como *data-driven*, *keyword-driven*, etc., nas quais combinam técnicas data-driven com a possibilidade de especificar os casos de teste de forma menos detalhada, tal como é feito quando estamos descrevendo um caso de teste manualmente.

A metodologia proposta neste trabalho visa explorar ferramentas do tipo capture-replay, como a TestComplete para apoiar a execução de testes funcionais na linha de produção de sistemas computacionais.

O principal motivo que levou o desenvolvimento desta pesquisa é o estabelecimento de padrões para cobertura de testes funcionais de forma automática, proporcionando a redução de defeitos causados por falhas no processo de teste.

1.4 ESTRUTURA DO TRABALHO

Quanto à estrutura formal, este trabalho divide-se em seis capítulos.

No primeiro deles, é contextualizada a introdução referente ao trabalho, são expostas as definições dos objetivos geral e específicos, sendo concluído com a apresentação da justificativa do trabalho realizado.

O segundo capítulo trata da Engenharia de Software e discorre sobre os principais padrões de qualidade do processo de software, juntamente com o modelo MPS.BR e as áreas do processo de teste.

No capitulo três, procura-se apresentar por detalhado os testes de software juntamente com seus tipos e técnicas existentes e mais utilizadas.

O quarto capítulo apresenta o estudo da automatização de teste juntamente com a especificação das técnicas e a apresentação da ferramenta *TestComplete.*

A apresentação dos trabalhos correlatos está descrita no quinto capítulo, e por fim, o capítulo seis exibe a metodologia e a definição do processo proposto, juntamente com toda a criação da metodologia do trabalho.

2ENGENHARIA DE SOFTWARE

De acordo com o conceito de engenharia, podemos afirmar que é a atividade onde os conhecimentos científicos, técnicos e a experiência são utilizados na prática para a exploração de recursos naturais e intelectuais gerando a construção e operação de objetos úteis.

A Engenharia de software, por sua vez, é uma área do conhecimento da computação voltada para a especificação, desenvolvimento e manutenção de sistemas de softwareaplicando tecnologias e práticas de gerência de projetos, objetivando organização, produtividade e qualidade(PRESSMAN, 2006).

A construção de sistemas computacionais deve obedecer a características de qualidade como flexibilidade, modularização, robustez, confiabilidade e usabilidade. Características estasque, devido as constantes alterações nos projetos de software, tornam-se difíceis de serem mantidas.

As constantes alterações do software ao longo do seu ciclo de desenvolvimento dificultam consideravelmente a aplicação das técnicas de gerenciamento de projetos e de garantia da qualidade. Desta forma, é de vital importância que testes sejam aplicados para que se possa garantir através da execução controlada do software, se seu comportamento está de acordo com o que se foi proposto.

A realização de testes origina uma ampla quantidade de informações úteis, ou seja, revela as falhas e identifica as suas causas para que possam ser corrigidas a fim de minimizar os problemas no produto e para que este seja distribuído ao mercado sem erros.

Segundo Pressman (2006), todo o desenvolvimento estaria constituído dentro das fases de definição, desenvolvimento e manutenção. Esta disciplina adota métodos, técnicas e princípios que visam facilitar o processo de desenvolvimento de sistemas tal que, este processo possa ser desempenhado de maneira analítica e, ao mesmo tempo, criativa.

A alta complexidade inferida nos sistemas computacionais encontrados no contexto atual tem fomentado o surgimento de diversas técnicas e metodologias que apóiam o controle das questões críticas da engenharia. Tomando por base que a qualidade do sistema desenvolvido esta diretamente relacionada com a qualidade do processo ao qual ele é submetido, podemos considerar que uma forma bastante

aceita no mercado para garantir a qualidade do desenvolvimento de sistemas baseados em processos complexos, é a adoção de um modelo de qualidade.

Ao longo dos anos já são encontrados no mercado, diversos modelos de ciclo de vida de desenvolvimento que, quando implementados, apóiam a construção de metodologias aplicadas à garantia da qualidade das soluções desenvolvidas. Dentre os modelos mais clássicos existentes, é possível destacar o modelo cascata e o modelo espiral, que, implementam na sua fase de construção os conceitos relacionados a metodologia de testes de software.

O ciclo de vida cascata é um dos modelos mais conhecidos pela engenharia de software. Nele as fases do projeto são executadas em uma sequência ordenada, onde a próxima etapa só poderá ser iniciada quando a etapa anterior estiver totalmente completa. Segundo Sommerville (2007) na prática, esses estágios se sobrepõem e trocam informações entre si.

O ciclo de vida espiral separa o projeto em pequenos projetos, e é orientado a riscos, ou seja, é possível prever e eliminar os problemas de grandes proporções por meio de projetos gerenciados e de um planejamento bem estruturado. Cada passo deste ciclo inclui planejamento, análise e projeto, prototipação e avaliação.

Os aspectos relacionados à validação das funcionalidades implementadas, estão implícitos nas fases da cada ciclo de vida, tendo obrigatoriamente que ser executados para que o sistema desenvolvido adquira um nível satisfatório de qualidade.

A apuração do nível de qualidade dos sistemas em relação a validação de suas funcionalidades, não é uma tarefa fácil e exata, possuindo uma forte dependência do conhecimento tácito das pessoas envolvidas no projeto.

A automação de testes de software minimiza consideravelmente a dependência do conhecimento tácito dos especialistas, uma vez que, o conhecimento fica registrado nos *scripts* de validação das estruturas funcionais dos sistemas propostos.

Hoje a qualidade é obrigatoriedade nos sistemas e as empresas estão se dedicando a desenvolver seus produtos seguindo conceitos e padrões de qualidade.

2.1 QUALIDADE DE SOFTWARE

Nos últimos anos, com o avanço crescente da tecnologia, o desenvolvimento de um software vem se tornando cada vez mais necessário em um ambiente coordenado. Com isso, o padrão de qualidade deste software vem se tornando cada vez mais indispensável. Este capítulo aborda os principais modelos e características que garantem a qualidade de um software.

A definição de qualidade para Pezzé e Young (2008) é um conceito abrangente que se realiza por intermédio de um conjunto de atividades interdependentes.Rocha, Maldonado e Weber (2001)ressaltam que a qualidade de software é, portanto, um conjunto de propriedades a serem satisfeitas, em determinado grau, de modo que o software satisfaça as necessidades de seus usuários. Já Pressman (2006) define qualidade de software como o conjunto de conformidades com requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas, que são esperadas em todo software desenvolvido profissionalmente.

De acordo com os conceitos apresentados, podemos afirmar que qualidade de software é um conceito que abrange os aspectos tanto das organizações quanto dos usuários e clientes, e que envolve múltiplas características que devem ser incorporadas em níveis diferentes do processo de desenvolvimento de um software, dentre elas a produtividade, usabilidade, eficiência, confiabilidade, portabilidade, segurança, manutenibilidade.

Assegurar a qualidade de um software, segundo Paula Filho (2001), é uma atividade que previne e visa minimizar o número de defeitos e falhas, e consequentemente, os problemas com o produto.

Pezzé e Young (2008), dizem em resumo que o processo da qualidade possui três objetivos:

a) melhorar um produto de software prevenindo, detectando e removendo erros;
b) avaliar a qualidade de um produto de software em relação às metas definidas de qualidade;
c) melhorar a longo prazo a qualidade e rentabilidade do processo da qualidade.

Bastos et al (2007) descreve os objetivos da garantia de qualidade como:

a) ajudar a estabelecer processos e os aprimorar;

b) determinar programas de medida para examinar processos;

c) identificarfragilidades em processos;

d) associadacom todos os produtos que serão gerados por um processo;

e) avaliar se o controle de qualidade está funcionando.

Pressman (2006) diz que qualidade é uma junção de diversos fatores que tendem a variar de acordo com as distintas aplicações e clientes. Essas variações são abordadas também por Rocha, Maldonado e Weber (2001) dizendo que o essencial é o software ser confiável, seguindo os padrões onde irá atuar.

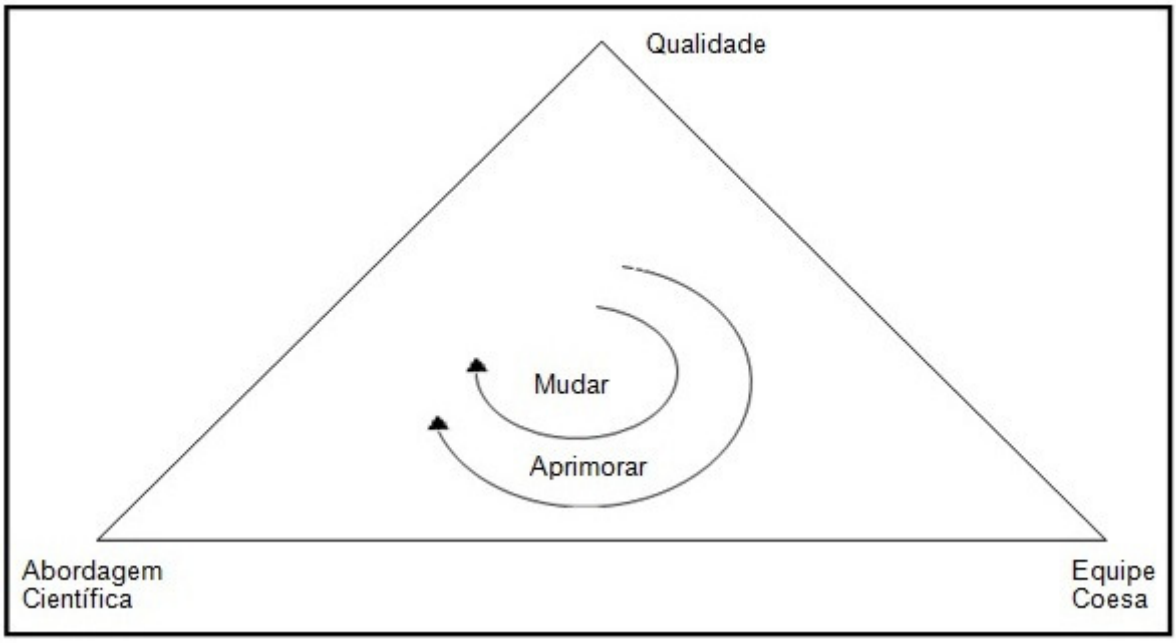
Sommerville (2007) destaca que o processo de garantia de qualidade está relacionado à definição e escolha dos padrões aplicados ao desenvolvimento ou ao produto de software. Rocha, Maldonado e Weber (2001) dizem ainda que paralelamente ao processo de qualidade devem estar os processos de verificação, validação, revisão e auditoria.

Sommerville (2007) destaca que a escolha dos métodos a serem utilizados no processo de produção ligados à qualidade é gerencial. Além disso, é importante ressaltar que a produção de software é um processo que envolve, principalmente, seres humanos. Assim, mesmo com tecnologias de produção de alta qualidade, de nada adianta se a equipe não apresentar qualidade equivalente.

“A qualidade deve estar presente não só nos produtos produzidos, como também nos processos utilizados para gerar esses produtos” (ROCHA; MALDONADO; WEBER, 2001, p. 242). Segundo os mesmos para esse caso, os processos de qualidade se aplicam aos produtos e aos próprios processos. Para ele a gerência é quem assegura a qualidade, mas é preciso por parte dela o conhecimento técnico sobre os métodos e ferramentas, além do conhecimento para lidar com as pessoas.

Rocha, Maldonado e Weber (2001) apresentam um modelo de gerência para aprimorar e satisfazer situações do dia-a-dia. Esse modelo é representado por um triângulo, e seus vértices representam qualidade, abordagem científica e coesão da equipe, como pode ser observado na figura 1.

Figura 1 -Triângulo de Joiner



Fonte: ROCHA; MALDONADO; WEBER (2001, p. 98)

O vértice superior do triângulo representa a qualidade do software definida pelo cliente, ou seja, aquilo que o cliente define e espera que o software realize. Ela está apoiada sobre os vértices de abordagem científica, representando processos racionais, métodos eficazes e ferramentas adequadas utilizados no projeto, e equipe coesa representando a equipe do projeto, uma equipe que conhece e busca atingir as metas e objetivos, para que todos saiam ganhando.

Neste modelo é analisada uma determinada situação pelo gerente em conjunto com a equipe. Caso a abordagem contribua para o aprimoramento de algum dos vértices, ela é aceita, caso contrário, ela é descartada. O objetivo é a evolução constante da equipe e o comprometimento das pessoas com o projeto, para assim, garantir a qualidade do software

Algumas instituições nacionais e internacionais desenvolvem esses padrões, como o Software EngineeringInstitute (SEI) do Carnegie MellonUniversity – Estados Unidos, InternationalOrganization for StandardizationandtheInternationalElectrotechnicalCommission (ISO/IEC) onde diversas normas e modelos de qualidade de processo de software foram propostas, dentre elas: CMMI, ISO/IEC 15504, ISO 9001,entre outros. Essas normas e modelos de qualidade tem por objetivo, apontar características que um processo de software deve apresentar, deixando a organização livre para sistematizá-lassegundo sua própria cultura.

Segundo o SEI, o CapabilityMaturityModelIntegration (CMMI) é uma abordagem de melhoria de processos que fornece às organizações os elementos de processos efetivos, o que melhorará o seu desempenho. Portanto, cada organização, de acordo com sua diretriz, pode implantar seus processos e atividades de desenvolvimento de software, seguindo os comandos do modelo.

Um processo pode ser definido como um conjunto de métodos, atividades, mudanças e práticas que deve ser usado para desenvolver e manter produtos e serviços. Todos os processos existentes mostram o quanto uma organização é madura para desenvolver e manter os produtos e serviços.

Segundo o CMMI, existem três tipos de análises sobre os processos organizacionais:

a) capacidade: Essa análise descreve a gama de resultados que podem ser atingidos com o uso dos processos;
b) desempenho: Analisa o atual estágio dos processos e os resultados obtidos pelo seu uso;
c) maturidade: Analisa até que ponto um processo específico está definido, gerenciado, mensurado, controlado e é efetivo. Maturidade implica ter potencial para um crescimento consistente aplicando os processos de software em projetos da organização.

O CMMI exige que haja planejamento e acompanhamento das atividades do processo de desenvolvimento do produto, através da adoção de métodos e modelos de ciclo de vida que guiem as atividades.

O tratamento por estágios do CMMI possui cinco níveis distintos de maturidade:

a) Nível 1 – Nível onde as organizações não possuem um ambiente estável, onde em geral seus colaboradores desempenham atos heróicos;
b) Nível 2 – Nível onde se há requisitos e processos, e os produtos e serviços são controlados;
c) Nível 3 – É caracterizado por processos bem definidos compreendidos aonde são descritos padrões nos procedimentos, nas ferramentas e nos métodos;
d) Nível 4 – Possui instrumentos para validação estatística dos processos, que é compreendido durante o desenvolvimento do processo,

baseando-se nas necessidades dos clientes, usuários, da organização como um todo;

e) Nivel 5 – Tem como objetivo a melhoria continua dos processos, onde são baseadas na compreensão do retorno previsto pela organização e pelo impacto que a mudança ocasionará.

O Spice (ISO/IEC 15504) é um modelo que, como o CMMI, possui seu foco na melhoria dos processos de desenvolvimento de software. O Spice prega práticas fundamentais para uma boa engenharia de software. A sua arquitetura de framework organiza essas praticas em duas distintas abordagens:

a) Práticas com foco em atividades importantes para processos específicos reunidos por tipo de atividade;
b) Praticas genéricas que representam fundamentalmente as atividades necessárias para a gestão do processo.

Para cada etapa do desenvolvimento e de manutenção de um produto de software o Spice possui processos bem definidos, onde são delineadas uma serie de atividades para a conclusão de cada processo. Os processos definidos pelo Spice são:

a) Cliente-Fornecedor, onde fundamentados os processos que possuem relação direta com o cliente;
b) A engenharia que mantém a especificação, execução, documentação e manutenção do software;
c) O projeto, que centraliza os recursos do mesmo;
d) O suporte que possui ênfase no melhor desempenho de sustentação para o projeto;
e) A organização que controla os processos que constituem os objetivos de negocio da organização, desenvolvendo processos, produtos e recursos.

Ao contrário do CMMI, o Spice possui seis níveis de capacidade (de 0 a 5) de uma organização, conforme segue:

a) Nível 0 – Não executado: trata-se da organização que não possui nenhum tipo de processo;
b) Nível 1 – Informalmente executado: trata-se da organização que possui algumas praticas básicas não documentadas e sem planejamento

rigoroso, onde na maioria das vezes o êxito em seus projetos deve-se aos esforços dos colaboradores;

c) Nivel 2 - Planejamento e rastreamento: os produtos apresentam uma especificação padrão e os requisitos começam a ser bem delineados;
d) Nivel 3 - Bem definido: usando processos padronizados, neste nível a organização já possui um planejamento e uma organização;
e) Nivel 4 – Controlado quantitativamente: neste nível a organização já possui ferramentas para a medição da melhoria do processo tendo seu desempenho gerenciado;
f) Nivel 5 – Melhoria continua: a organização implementa novas idéias utilizando inovações tecnológicas e possui um nível avançado de controle e gerenciamento que detecta falhas em seus processos, podendo assim corrigi-los.

Uma organização pode ser considerada do nível 2 quando todos os atributos dos níveis inferiores são totalmente atendidos e todos os atributos do nível são amplamente atendidos.

2.2MODELO DE QUALIDADE MPS.BR

O MPS.BR é um modelo de qualidade que tem em vista omelhoramentodo processo do software brasileiro. A descrição geral deste modelo e do modelo de referência (MR-MPS), e os significados necessários para o seu entendimento e aplicação, são encontrados neste modelo.

Criado em dezembro de 2003 o MPS.BR é um programa estimulador, coordenado pela Associação para Promoção da Excelência do Software Brasileiro (SOFTEX), que conta com apoio do Ministério da Ciência e Tecnologia (MCT).

Segundo a SOFTEX (2009a) oMPS.BR visa a Melhoria de Processo do Software Brasileiro e seus objetivos são:

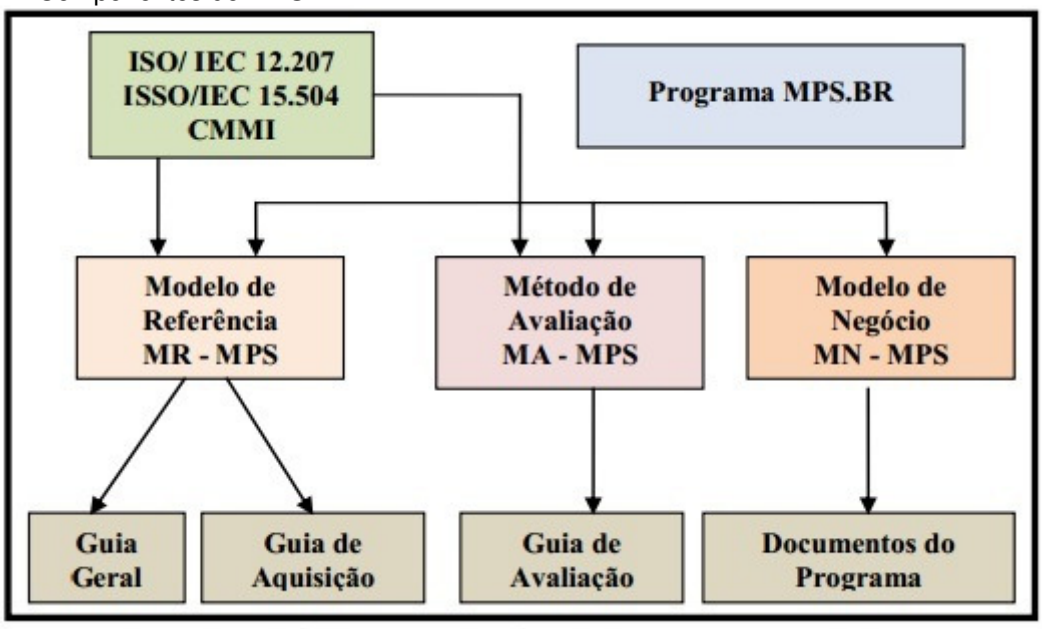
a) Definir um modelo de melhoria e avaliação de processo desoftware, visando às micros, pequenas e médias empresas,atendendo as suas necessidades de negócio;
b) Ser reconhecido nacionale internacionalmente como um modelo aplicável à indústria de software;

c) Adequar ao perfil de empresas com diferentes tamanhos e características,públicas e privadas;
d) Possuir compatibilidade com os padrões de qualidade aceitosinternacionalmente,presumindo o aproveitamento de toda acompetência existente nos padrões e modelos de melhoria de processo acessíveis.

Uma das suas metas é aprimorar um modelo de melhoria e avaliação do processo de software, tendo em vista as micros, pequenas e médias empresas, de forma a atender as suas necessidades de negócio e ser aprovado nacional e internacionalmente como um modelo aplicável a indústria de software.

Conforme SOFTEX (2009a) o MPS.BR está dividido em três componentes:Modelo de Referência (MR-MPS), Método de Avaliação (MA-MPS) e o Modelo de Negócio (MN-MPS), onde cada componente é exposto por meio de documentos ou guias, conforme figura 2. Os modelos de avaliações de processos garantem que o MPS.BR está sendo empregados de forma coerente com suas definições e metas.

Figura 2 - Componentes do MPS.BR



Fonte: SOFTEX (2009a)

O Modelo de Referência MR-MPS define níveis de maturidade que são umacombinação entre processos e sua capacidade. A definição dos processos segue os requisitospara um modelo de referência de processo apresentados na

ISO/IEC 15504-2, revelando opropósito e os resultados esperados de sua execução. Isso permite avaliar e atribuir graus deefetividade na execução dos processos.

O MR-MPS possui sete níveis de maturidade, sendo eles:

a) A - Em Otimização;
b) B - Gerenciado Quantitativamente;
c) C - Definido;
d) D - Largamente Definido;
e) E - Parcialmente Definido;
f) F - Gerenciado;
g) G - Parcialmente Gerenciado.

Cada nível de maturidade possui suas áreas de processos, nas quais os processos e atributos da norma ISO/IEC 12207 são analisados e indicam onde os esforços de melhoria da organização devem ser focados para que atinjam os objetivos do negócio e do modelo de referencia. (SOFTEX, 2009a).

A capacidade do processo no MPS-BR possui alguns atributos de processos que são:

a) AP1.1- O processo é realizado;
b) AP2.1 - O processo é gerenciado;
c) AP2.2 - Os produtos de trabalho do processo são gerenciados;
d) AP3.1 - O processo é determinado;
e) AP3.2 - O processo está desenvolvido;
f) AP4.1 - O processo é medido;
g) AP4.2 - O processo é controlado;
h) AP5.1 - O processo é foco de inovações;
i) AP5.2 O processo é otimizado continuamente.

Na Tabela 1 - Níveis de maturidade de processo doMPS.BRsãoevidenciados os processos abordados em cada nível doMPS.BR e suas respectivas características de processo.Cada nível é formado por algumas áreas de processo que são implementadas de acordo com um conjunto de resultados estabelecidos para cada atributo de processo. O método de avaliação do modelo MPS.BR tem o propósito de verificar a maturidade de uma organização no momento da execução de seus produtos de software.

O processo e o método de avaliação MA-MPS são definidos da seguinte forma:

a) Permitir a avaliação sucinta dos processos de software de uma unidade organizacional;
b) Permitir a atribuição de um nível de maturidade do MR-MPS com fundamento no resultado da avaliação;
c) Ser aplicável a qualquer domínio de aplicação na indústria de software;
d) Ser aplicável a unidades organizacionais de qualquer tamanho.

Tabela 1 - Níveis de maturidade de processo doMPS.BR

Nível	Processos	Atributos de Processo
A		AP 1.1, AP 2.1, AP 2.2, AP 3.1, AP 3.2, AP 4.1, AP 4.2 , AP 5.1 e AP 5.2
B	Gerência de Projetos – GPR (evolução)	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2, AP 4.1 e AP, 4.2
C	Gerência de Riscos – GRI	AP 1.1, AP 2.1, AP 2.2,AP 3.1 e AP 3.2
	Desenvolvimento para Reutilização – DRU	
	Gerência de Decisões – GDE	
D	Verificação – VER	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Validação – VAL	
	Projeto e Construção do Produto – PCP	
	Integração do Produto – ITP	
	Desenvolvimento de Requisitos – DRE	
E	Gerência de Projetos – GPR (evolução)	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Gerência de Reutilização – GRU	
	Gerência de Recursos Humanos – GRH	
	Definição do Processo Organizacional – DFP	
	Avaliação e Melhoria do Processo Organizacional - AMP	
F	Medição – MED	AP 1.1, AP 2.1 e AP 2.2
	Garantia da Qualidade – GQA	
	Gerência de Portfólio de Projetos – GPP	
	Gerência de Configuração – GCO	
	Aquisição – AQU	
G	Gerência de Requisitos (GRE):	AP 1.1 e AP 2.1
	Gerência de Projetos (GPR):	

Fonte: SOFTEX – (2009a)

2.3ÁREA DE PROCESSO

Segundo Sommerville (2007) um processo de software é um conjunto de atividades e resultados unidos, que levam a produção de um produto de software. Este processo pode envolver o desenvolvimento de software desde o inicio, embora ocorra o caso de um software novo ser desenvolvido mediante a expansão e a modificação de sistemas já existentes.

O mesmo autor afirma que os processos de software são difíceis e complexos e, como todos os processos intelectuais, dependem de julgamento humano. Existe uma imensa diversidade de processos de software, por este modo não há um processo ideal. Diferentes organizações desenvolveram tratamentos distintos para o desenvolvimento dos seus softwares.

Segundo Pressman (2006) os modelos de processo que enfatizam a definição, identificação e aplicação detalhada de atividades e tarefas de processo, também chamados de modelos prescritivos de processo, têm sido aplicados nas comunidades de engenharia de software durante os últimos trinta anos. Esses modelos têm como objetivo guiar equipes de engenheiros de software à medida que a construção do sistema seja feito, melhorando assim a qualidade do sistema para que os projetos se tornem mais gerenciáveis e que seu custo e seu prazo de entrega sejam mais previsíveis.

O autor também relata que os modelos de processo que ressaltam a agilidade do projeto e seguem uma série de princípios que levam uma abordagem mais informal, porém não menos efetivas, do processo de software foram propostos nos últimos anos. Estes modelos, também chamados de modelos de processo ágil, enfatizam a adaptabilidade e são adequados a muitos tipos de projetos.

Sommerville (2007) acrescenta que embora existam muitos processos diferentes de softwares, há atividades fundamentais que são comuns a todos eles, como:

a) especificação de software – É preciso que seja definido as funcionalidades do software e as restrições em sua operação;
b) projeto e implementação de software – As especificações devem ser cumpridas no momento do desenvolvimento do software;

c) validação de software – O software deve ser validado para que sua funcionalidade esteja de acordo com as exigências do cliente;
d) evolução do software – O software precisa evoluir para atender as necessidades do cliente.

Embora não exista nenhum processo de software ideal, Sommerville (2007) diz que melhorias dos processos podem ser implementadas de diferentes maneiras. A padronização do processo é uma primeira etapa essencial na introdução de novos métodos e novas técnicas de engenharia de software.

3TESTE DE SOFTWARE

Atualmente a usabilidade de softwares mostra-se fundamental no mercado empresarial, o que faz com que profissionais da área de Tecnologia da Informação(TI), se preocupem comerros existentes em seu produto, pois em uma organização inúmeras atividades dependem do seu correto funcionamento. Para tanto o presente capítulo aborda o teste de software, tipo de testes de software, técnicas e métodos de teste.

Processos de desenvolvimento e padrões aplicados a um software estão diretamente relacionados asua qualidade. O teste de software é uma importante etapa deste processo, cuja função é encontrar erros, certificando assim, a qualidade do programa testado.

Conforme Rios e Moreira (2006), nas décadas de 1960 e 1970 no processo de desenvolvimento a maior preocupação era em relação a codificação e não os testes. Nessa época os testes eram feitos pelos desenvolvedores e se comparam hoje com os testes unitários e de integração. Uma maior importânciapara os testes passou-se a se dar a partir da década de 80 quando este começou a ser tratado como processo formal do desenvolvimento.

Pezzè e Young (2008) definem o teste de software como uma atividade que tem por objetivo validar e verificar o produto, revelando defeitos e aumentando a sua confiabilidade.Molinari (2003), completa dizendo que além de encontrar erros o mais cedo possível, a função do teste é fazer com que eles sejam ajustados.

A função da fase de teste, segundo Braude (2005), é disponibilizar a entrada ao programa e verificar se a saída é correspondente ao que foi determinado pelos requisitos do software.

Os atributos de qualidades citados por Pressman (2006) são definidos da seguinte maneira:

a) Funcionalidade: é avaliada pela observação do conjunto de características e capacidades do programa, generalidade das funções entregues e segurança do sistema global;
b) Usabilidade: é avaliada considerando fatores humanos, estética, consistência e documentações globais;
c) Confiabilidade: é avaliada medindo-se a freqüência e a severidade das falhas, a precisão dos resultados de saída, o tempo médio entre falhas,

a capacidade de recuperação de falhas e a previsibilidade do programa;

d) Desempenho: é medido pela velocidade de processamento, tempo de resposta, consumo de recursos, vazão e eficiência;
e) Suportabilidade: combina a capacidade de estender o programa (extensibilidade), adaptabilidade, reparabilidade – esses três atributos representam em termo mais comum, manutenabilidade –, além de testabilidade, compatibilidade, configurabilidade, facilidade com a qualo sistema pode ser instalado e facilidade com a qual os problemas podem ser localizados.

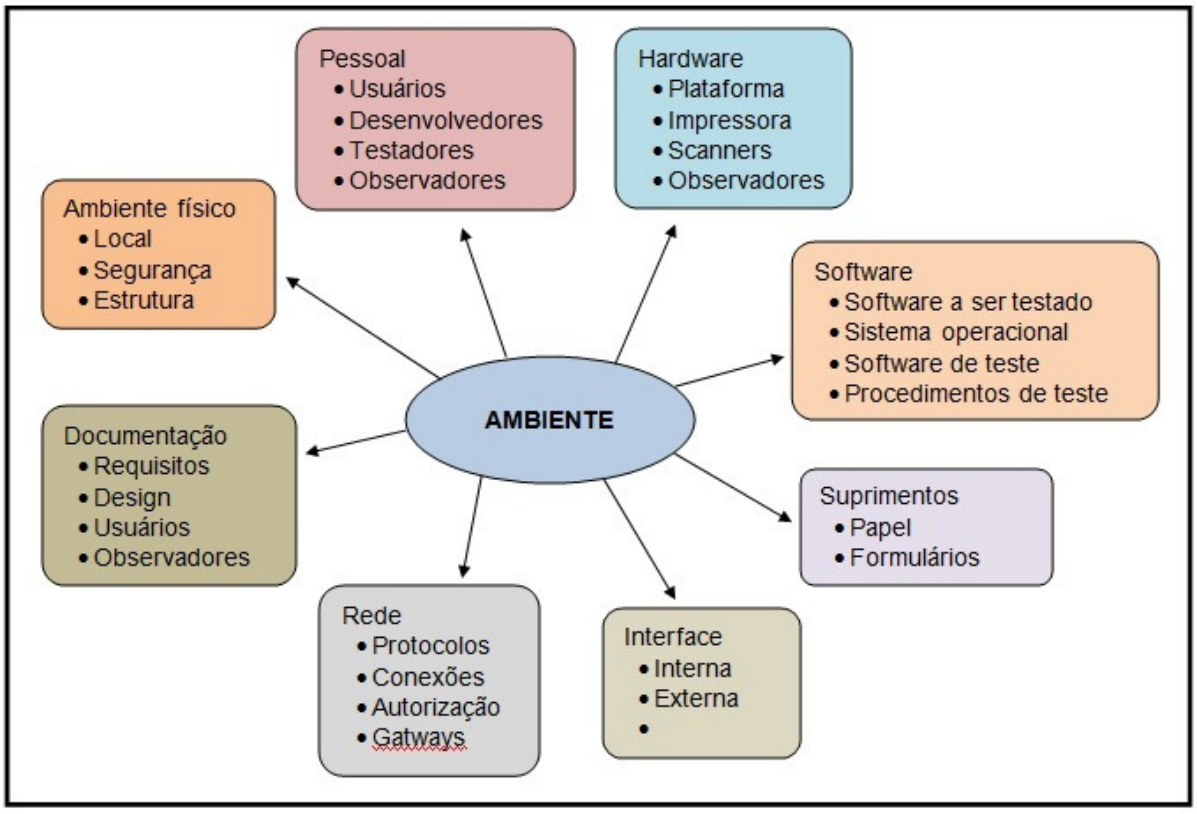
Paula Filho (2001) diz que defeitos são encontrados primeiro nas revisões onde são mais eficazes de encontrá-los e corrigi-los. A fase de teste é muito importante, pois alguns erros que podem passar desapercebidos pelas revisões, serão encontrados então nesta fase, garantindo a confiabilidade do software.

As práticas de teste, para Pezzè e Young (2008), começam antes mesmo que o código esteja finalizado. Para que erros surgidos nas fases iniciais não se difundam para as outras fases de desenvolvimento, as atividades de teste devem ser iniciadas assim que os elementos necessários para o teste estejam disponíveis.

Bastos et al. (2007), completa dizendo que também é necessário planejar o ambiente onde os testes serão realizados, criando uma massa de dados para o teste, o modelo de dados, a configuração dos softwares (*browsers*, sistemas operacionais, entre outros), enfim, toda a estrutura onde o teste será realizado, juntamente com as configurações do *hardware*. Esta composição é demonstrada na figura3.

Para Molinari (2003) o teste deve possuir planejamento (gerando um plano de teste), projeto dos casos de teste (situações possíveis de teste), execução e avaliação dos testes. Paula Filho (2001) também acredita que planejar e desenhar cuidadosamente os testes é necessário, e que essas atividades devem ser cumpridas por pessoas que tenham experiência para poderem analisar os resultados dos testes.

Figura 3 - Ambiente de Teste



Fonte: Bastos et al (2007, p. 77)

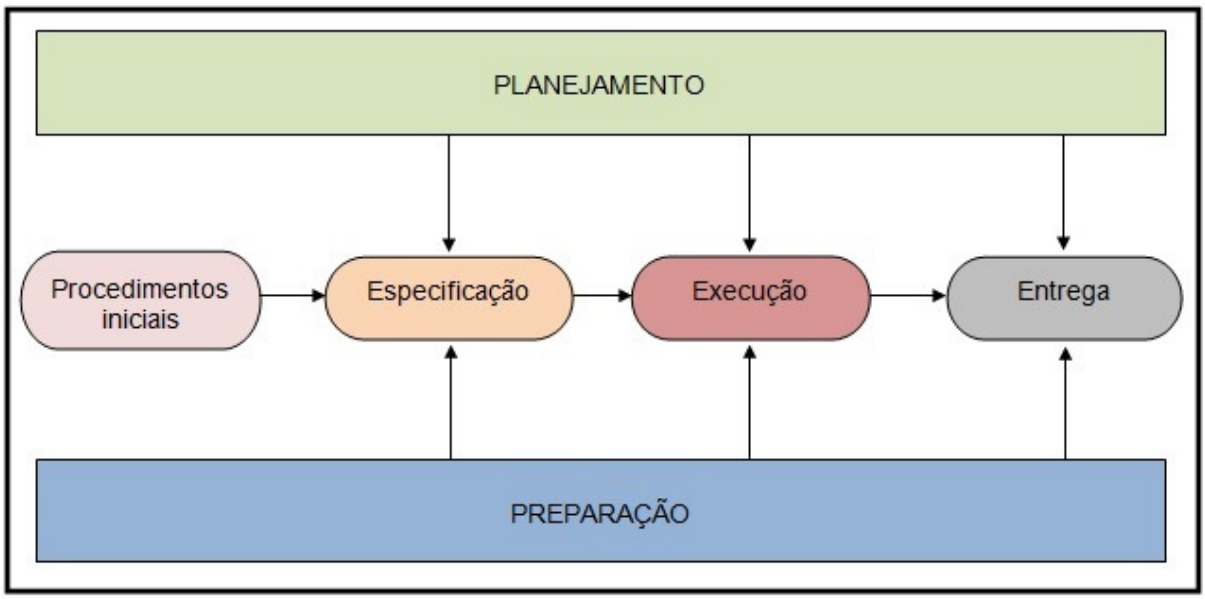
Para prevenir falhas no processo de teste, Rocha, Maldonado e Weber (2001) acreditam que é necessário modificar os níveis de teste em relação as etapas.

Rios e Moreira (2006) separam as etapas do processo de teste como: procedimentos iniciais, planejamento, preparação, especificação, execução e entrega. Essas fases podem ser vistas na figura4.

Rios e Moreira (2006, p. 8) também apresentam algumas técnicas de verificação usadas para encontrar defeitos:

> Dentro de um processo de testes existem também as técnicas de verificação, tais como: inspeção, revisão de produtos e "*walkthroughs*". Estas técnicas baseadas em reuniões e "*check-lists*" servem para identificar defeitos de elaboração, descumprimento de padrões e das boas práticas. Devem ser realizados em documentos produzidos, planos, códigos, especificações, requisitos, etc., preferencialmente antes da execução dos testes. Estas técnicas, usadas de forma combinada com os testes, aumentam sensivelmente a qualidade final dos softwares desenvolvidos.

Figura 4 - Etapas do processo de teste



Fonte: Rios e Moreira (2006, p. 9)

Delamaro, Maldonado e Jino (2007) dividem as atividades de teste em fases com objetivos distintos:

a) teste de unidade: Aponta erros de programação. Testa as menores unidades de um programa, podendo ser aplicada a cada nova implementação;
b) teste de integração: testa a estrutura do sistema, validando as interações efetuadas entre as partes do sistema e se as mesmas estão funcionando corretamente;
c) teste de sistemas: testa o sistema em um todo após seu termino, verifica se as funcionalidades estão de acordo com o que foi documentado.

Alguns autores como Delamaro, Maldonado e Jino (2007) destacam uma quarta fase de teste, chamando-a de teste de regressão. Com a função de verificar se as funcionalidades do sistema continuam executando suas tarefas de forma correta, esta fase é mais comumente realizada na manutenção do software.

Diversas técnicas de teste estão sendo aplicadas para encontrar erros, segundo Delamaro, Maldonado e Jino (2007). Porém para que haja benefícios para as empresas, é necessário utilizar uma ferramenta que faça o planejamento das atividades de testes.

3.1 TIPOS E TÉCNICAS DE TESTES DE SOFTWARE

Conforme mencionado anteriormente, para que se avalie a qualidade do software, é necessário a utilização de tipos e técnicas de teste. Tais técnicas diferenciam-se pela origem da informação empregada na avaliação e construção dos casos de teste.

Essas técnicas conforme Delamaro, Maldonado e Jino (2007) podem ser eficientes na descoberta de defeitos e na garantiada qualidade do software. Várias técnicas podem ser utilizadas para se administrar e avaliar a qualidade de teste.

Rezende (2005) afirma que em um mesmo projeto de sistema, vários tipos de testes podem ser empregados, porém, eles devem estar diretamente relacionados com as bases, o banco de dados, as características do software, com os recursos disponíveis, as fontes e com o tempo disponível para a realização dos testes.

Pezzè e Young (2008) completam dizendo que a escolha das técnicas que serão utilizadas no teste dependem dos recursos no desenvolvimento do produto, das restrições de qualidade, custo e prazo de entrega.

Novas técnicas de testes, ou melhor, tipos de testes, sempre irão surgir. Nada na vida é estático, porém sempre uma prática será mais eficiente do que a outra na maior parte dos ambientes, em particular, no seu ambiente de teste. (MOLINARI, 2003).

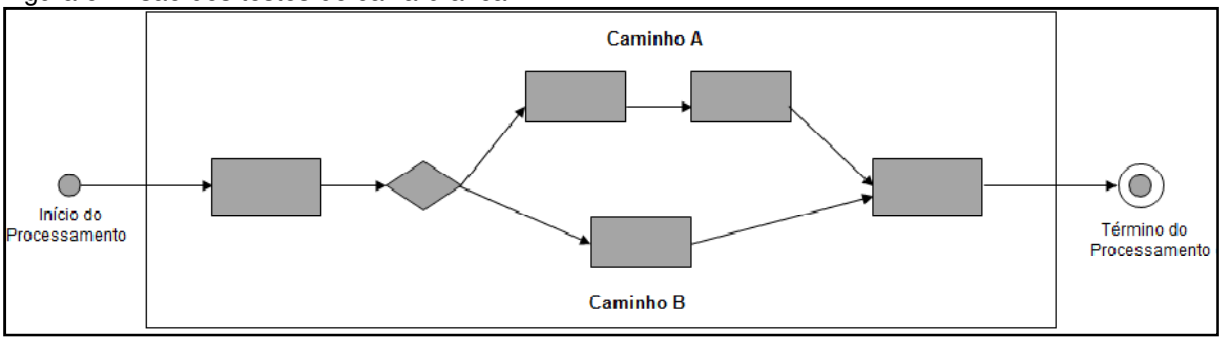
Paula Filho (2001) e Pressman (2006) classificam os testes em dois grupos principais: teste de caixa preta e teste de caixa branca.

3.1.1 Teste estrutural(caixa branca)

Também conhecido como teste estrutural, o teste de caixa branca avalia diretamente o código-fonte do componente de software para avaliar alguns aspectos, tais como: teste de condição, teste de fluxo de dados, teste de ciclos e teste de caminhos lógicos.

Bartié (2002) explica que os testes de caixa branca se fundamentam na arquitetura interna do software onde são aplicadas técnicas de testes para localizar defeitos e verificar todas as estruturas utilizadas na codificação. Esse tipo de teste percorre os caminhos internos da estrutura. Como pode ser visto na figura 5.

Figura 5 -Visão dos testes de caixa branca



Fonte: Bartié (2002, p. 104)

Esses tipos de teste, segundo Rios e Moreira (2006) visam avaliar diversas partes do código, a lógica interna do elemento codificado, as configurações e outros elementos técnicos. É considerado para Delamaro, Maldonado e Jino (2007) um complemento para as demais técnicas.

Bastos et al (2007), acredita que devido a necessidade de verificar se os sistemas funcionavam corretamente, a técnica do teste estrutural foi criada. Delamaro, Maldonado e Jino (2007), ressaltam que no princípio os testes estruturais se baseavam no fluxo de controle dos programas.

Esta técnica para Pressman (2006) possui quatro benefícios:

a) assegura que dentro de um módulo todos os caminhos tenham sido executados ao menos uma vez;
b) todos os valores falsos ou verdadeiros, decisões lógicas, sejam executados;
c) todos os laços sejam executados;
d) garantem a validade das estruturas de dados internas, as executando.

3.1.2 Teste funcional (caixa preta)

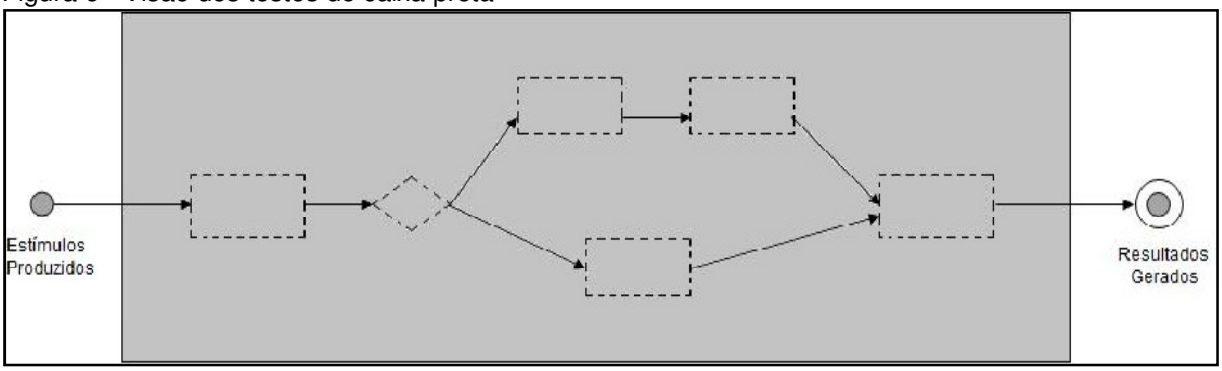
O teste de caixa preta, também conhecido como teste funcional, não considera o comportamento interno do componente de software a ser testando. Neste teste dados de entrada são fornecidos e o teste é executando, gerando um resultado onde o mesmo é comparado a um resultado esperado previamente conhecido. O teste só obterá sucesso caso o resultado alcançado for igual ao resultado esperado.

Para Rios e Moreira (2006), essa técnica de teste conferea funcionalidade do sistema do ponto de vista do usuário, sem considerar a lógica interna do elemento a ser testado ou o código, pois não há conhecimento sobre a operação interna do mesmo.

Esta técnica, segundo Molinari (2003), Bartié (2002), Pressman (2006) e Delamaro, Maldonado e Jino (2007), foca-se nos requisitos funcionais do sistema. São determinadas, a partir da especificação, as saídas esperadas para certos conjuntos de entrada de dados. Conforme pode ser verificado na figura6.

Bartié (2002) afirma que o grande benefício deste método é não necessitar ter o conhecimento sobre os códigos fontes utilizado para o desenvolvimento do software ou entender sobre a tecnologia aplicada sobre o mesmo. Para a execução dos testes é necessário apenas conhecer os requisitos básicos e verificar se a especificação está sendo atendida pelo software, o que torna mais fácil a busca por profissionais.

Figura 6 - Visão dos testes de caixa preta



Fonte: Bartié (2002, p. 104)

Conforme Delamaro, Maldonado e Jino (2007) nos anos 70, métodos para descrever os requisitos e apoiar a especificação de sistemas foram incorporados a técnica de teste funcional. Pressman (2006) e Molinari (2003) dizem que testes funcionais procuram encontrar basicamente os seguintes erros:

a) De interface;
b) No acesso a bancos de dados e nas estruturas de dados;
c) De desempenho;
d) De inicialização e término;
e) Funções ausentes ou incorretas.

Completa Pfleeger (2004) que o teste funcional deve ter algumas características, tais como:

a) Detecção grande de erros;
b) Conhecimento das ações e saídas esperadas;
c) Uma equipe de teste independente dos projetistas e programadores;
d) Testes de entradas validas e invalidas;
e) Possuir um critério de encerramento.

3.2 GERAÇÃO DA MASSA DE TESTES FUNCIONAL

Conforme Bastos (2007) os casos de testes são organizados com base em três documentos:

a) especificação de projeto de teste - Documento que identifica as funcionalidades e as características a serem testadas;
b) especificação de caso de teste - Documento que define os casos de testes, dados de entrada e resultados esperados;
c) especificação do procedimento de teste - Documento que aponta todos os passos necessários para a execução dos casos de teste especificados.

Um caso de teste estabelece quais informações serão empregadas durante o teste doscenários e quais serão os resultados esperados. Para que um caso de teste seja completo é necessário possuir os seguintes elementos:

a) identificação das condições de testes: pré-condições, pós-condições e critério de aceitação;
b) identificação dos casos de teste;
c) identificação do autor;
d) massa de dados de entrada e saída;
e) configuração do ambiente de teste;
f) identificação da versão da aplicação a ser testada;
g) descrição das ferramentas necessárias (sistema operacional, browser, origem dos dados etc.);
h) definição do modo como o teste será executado (manual ou automático);

i) identificação dos responsáveis pela elaboração e execução do caso teste;
j) definição da fase na qual o teste será executado (integração, sistema, regressão etc.);
k) listagem das dependências, caso existam, com outros casos de teste.

Vários métodos para a geração dos cenários de testes são disponíveis para uso conforme Bastos (2007), dentre eles o método passo-a-passo *(step-by-step)* é o mais utilizado em testes funcionais. Este método tem como objetivo produzir instantaneamente casos de testes completos para a especificação do sistema, de acordo com os seguintes passos:

a) Adicionar requisitos de teste garantindo adequadamente o domínio de cada entrada.Consistem em testar valores-limite, valor médio, condições de erros e entradas inválidas;
b) Listar os requisitos de teste de acordo com as especificações do sistema e gerar uma lista de testes, no objetivo dedelinear um escopo a ser testado;
c) Identificar um caso para cada requisito de teste e analisar as várias perspectivaspara que seja possível detectar diferentes tipos de defeitos.
d) Para cada caso de teste, considerar as entradas e saídas e asconfigurações necessárias para a execução.
e) Agrupar os casos de teste com entradas e configurações comum em suítes de teste.
f) Revisar os casos de teste, completando o que for necessário.

3.2.1 Derivação baseada no valor limite

A análise do valor limite (*BoundaryValueAnalysis*- BVA), conforme Rocha, Maldonado e Weber (2001), é um processo que completa o método de particionamento de equivalência, testando os limites de cada classe de equivalênciasendo que os limites associados às condições de entrada são exercitados de forma mais rigorosa.

De acordo com Meyers (2004), a experiência mostra que casos de teste que exploram condições limites têm maior possibilidade de encontrar defeitos.

Estascondições correspondem a valores que estão exatamente sobre ou imediatamente acima ou abaixo dos limitantes das classes de equivalência.

Para Meyers (2004) outro ponto que diferencia o valor limite do particionamento de equivalência é a observação do domínio de saída, além da escolha seletiva dos dados de teste.

Pressman (2006) e Molinari (2003) distinguem algumas diretrizes desse método:

a) Se especificado uma condição de entrada, com intervalo de valores entre a e b, o teste deve ocorrer com valores acima e abaixo dos limites desse intervalo.
b) Se especificado uma condição, com uma série de valores, o teste deve ocorrer com valores acima e abaixo do valor mínimo e máximo.
c) Se estruturas internas de dados do programa tiverem prescrito fronteiras, deve ser testado à estrutura de dados em sua fronteira.

Cada valor inválido ou válido, segundo Pressman (2006), deve ser representado em ao menos um caso de teste. Este processo gera duas vezes mais testes se comparado com a técnica de partição por equivalência, pois há dois valores limites para cada partição. A sua análise verifica se o espaço delimitado entre uma partição e outra está correto.

Delamaro, Maldonado e Jino (2007) julgam as vantagens e desvantagens deste critério similares ao critério de particionamento de equivalência, sendo que existem diretrizes para que os dados de teste sejam estabelecidos, uma vez que estes devem explorar especificamente os limites das classes identificadas.

4 TESTES AUTOMATIZADOS

A automação de testes é um processo de testes híbrido, já que testes manuais por hábito são indispensáveis. Os testes manuais consistem em testes onde cada passo é descrito detalhadamente. Sua execução é feita de forma que o profissional de teste simule um usuário comum interagindo direto com a aplicação (informando os dados manualmente, clicando nos botões etc.).

A execução manual de um caso de teste se torna uma função rápida, porém a execução e repetição de um inúmero conjunto de testes é uma tarefa muito dispendiosa e cansativa.

Neste contexto é imprescindível a utilização de ferramentas de automação de execução de testes. Segundo Molinari (2010) os testes automatizados utilizam uma ferramenta que copia ainteração com a aplicação tal qual um humano faria, porém com algumas limitações. A ferramenta chamaos *scripts* de testes que já estão previamente criados e os executam. Estes*scripts*que contêm os dados deentrada e as ações que deverão ser executadas sem interferência do profissional de testes, ouseja, testessignificativos do caso de teste não serão ignorados por falha humana e facilitarão aidentificação de um possível comportamento não desejado.

Por se tratar de um trabalho maçante, repetitivo e que exige tempo e organização por parte dos testadores, as organizações estão investindo na automatização dos testes, utilizando ferramentas apropriadas para tal função. Com a utilização destas ferramentas, pode-se identificar erros com mais eficiência. A tabela 2 apresenta uma comparação entre o teste manual e o teste automatizado de 1.750 casos de testes com 700 defeitos existentes.

Tabela 2 - Comparativo em horas entre os testes manuais e automatizados

etapas dos testes	teste manual	teste automatizado	melhoria (%)
Planejamento	32	40	-25 %
definição de casos de testes	262	117	55 %
execução dos testes	466	23	95 %
conferência dos testes	117	58	50 %
gerenciamento do erro	117	23	80 %
relatórios finais	96	16	83 %
duração total (em horas)	1.090	277	75 %

Fonte: Bartié (2002, p. 64)

Através da análise desta tabela, Bartié (2002) afirma que os testes automatizados ao logo do seu tempo de uso são mais econômicos, rápidos e eficientes do que os testes manuais. Acredita-se, segundo Moreira Filho e Rios (2003), que a automatização de testes significa a substituição do testador. No entanto, a automatização é apenas uma maneira de tornar os testes mais eficientes e confiáveis, representando um apoio à sua execução, pois é fundamental o papel do testador que cria e atualiza os *scripts*. Com a automatização, os testes tornam-se mais precisos e encontram-se rapidamente as falhas, pois os *scripts* podem ser rodados a qualquer momento, repetindo os mesmos testes, algo que seria um trabalho cansativopara o testador.

Existem várias ferramentas de execução automática de testes, cada uma se que se adapta melhor ao tipo de teste a ser automatizado. O sucesso da automação de testes está diretamente ligado com a documentação dos casos de testes que serão automatizados e com as ferramentas que serão utilizadas no processo de automatização.

Neste trabalho, dentre as ferramentas de automação de teste existentes, será feita a utilização do TestComplete.

4.1 FERRAMENTA TESTCOMPLETE

Segundo a AutomatedQA Corporation, o TestComplete é uma ferramenta utilizada para automação de testes para aplicações Windows e .NET, porém mesmo que a ferramenta seja uma aplicação Windows, tanto o sistema operacional quanto o tipo do servidor web testado não importam, pois a ferramenta suporta servidores de diversas plataformas (Linux, Windows, etc.).

A escolha desta ferramenta se deu, pois a mesma não depende de qualquer outra ferramenta de desenvolvimento.Obtém da gravação de uma série de comandos que simulam as ações do utilizador, a integração com os navegadores é de fácil execução e principalmente, pois esta ferramenta apóia plenamente *front-end* de testes web garantindo que a funcionalidade e a confiabilidade dos sites e aplicações que irão ao ar.

O TestCompleteé aplicado a testes de unidade, funcional e de regressão. As principais funcionalidades desta ferramenta são:

- grava a interação do usuário com a interface do software, através dos eventos do mouse e teclado;
- permite criar e alterar um *script* de teste;
- reconhece variáveis e componentes da interface;
- permite gerar o resultado dos testes em arquivo de extensão
- permite depurar o *script* incluindo breakpoints;
- realiza testes em interfaces desktop e web.

Pressman (2006) ressalta que o aumento das exigências dos usuários e da concorrência está aumentando cada vez mais na área de desenvolvimento de software, o que faz com que as empresas queiram aperfeiçoar seus processos. No entanto, considerando a complexidade envolvida nos sistemas atuais, para se testar adequadamente um software, uma organização gasta 40% do esforço de projeto total em teste. Para diminuir o tempo e custo do processo de teste de software esta ferramenta propõem auxiliar os colaborados nas atividades relacionadas a este processo, utilizando as técnicas existentes de automação de teste que será abordado logo abaixo.

4.2 – TÉCNICAS DE AUTOMAÇÃO DE TESTE

Os testes funcionais, de acordo com Molinari (2003) podem ser subdivididos em dois diferentes paradigmas dentro do processo de automação:

a) baseados na Interface Gráfica: Os testes interagem diretamente com a interface gráfica da aplicação, simulando um usuário.
b) baseados na Lógica de Negócio: Nesta abordagem os testes automatizados praticam as funcionalidades da aplicação, não interagindo com a interface gráfica.

Os tipos mais conhecidos de automação de testes, segundo Molinari (2003) são:

a) testes automatizados baseados na interface gráfica (*Capture/Playback):* Esta arquitetura de automação é uma das mais simples. Nela são utilizados somente os recursos da ferramenta, carturando (Cpture) e gravando uma ação humana interagindo com o

aplicativo e reproduzindo (Playback) quando for solicitado, os dados de entrada ficam armazenados em um script;

b) testes automatizados dirigidos a dados *(Data-Driven):* Esta arquitetura é mais evoluída, pois possui um alto grau de reutilização e conseqüentemente diminui a complexidade. Utiliza o mecanismo na qual os dados de entrada do aplicativo são isolados do *script*, em seguida os dados são retirados do *script* e armazenados em uma estrutura de dados externa. Este processo na execução de testes que executam as mesmas ações repetidamente, porém utilizando dados de entrada diferentes;

c) testes automatizados dirigidos à palavra-chave (*Keyword-Driven)*: Esta arquitetura é considerada a mais avançada e exige um alto investimento para implementação. Os testes são baseados em palavras-chaves (keywords), ou seja, para a criação dos testes a ferramente de automação monta um conjunto pré-definido de palavras-chaves. Cada palavra-chave é um comando em alto nível que representa uma ação. Esta abordagem foi criada para dar suporte aos testes de aceitação *(AcceptanceTests);*

d) testes automatizados baseados na linha de comando (CLI):Uma Interface de Linha de Comando permite que o usuário possa interagir com aplicação por meio de um interpretador de linha de comando do sistema operacional.

Os tipos acima citados são normalmente agrupados de acordo com a forma como os testes automatizados interagem com a aplicação.

5Trabalhos correlatos

Foram analisados alguns trabalhos correlatos envolvendo assuntos importantes como as ferramentas de automatização de teste e ferramentas de apoio a definição de processos.

5.1 DEF-PRO: APOIO AUTOMATIZADO PARA A DEFINIÇÃO DE PROCESSOS DE SOFTWARE

Este artigo foi desenvolvido por Luis Filipe D. Cavalcanti Machado, Gleison Santos,Káthia Marçal de Oliveira, Ana Regina Cavalcanti da Rocha, sendo apresentado na Universidade Federal do Rio de Janeiro na tarefa de apoiar a definição do processo de *software* padrão e uma organização iniciando-se pela definição do processo padrão.

Segundo Machado, Santos, Oliveira e Rocha, a ferramenta Def-Pro tem como objetivo a definição de um processo de *software* padrão para uma organização com base na Norma ISO/IEC 12207, nos modelos de maturidade, em níveis de capacitação, nas características do desenvolvimento de *software* da organização e no tipo de ADS para o qual se está definindo o processo. Esta ferramenta de fácil manuseio é de grande utilidade no apoio àdefinição de processos padrões adequados às organizações e aospadrões internacionais.

5.2 AUTOMAÇÃO DE TESTES UTILIZANDO AS FERRAMENTAS RATIONAL QUALITY MANAGER E RATIONAL FUNCTIONAL TESTER

No trabalho de conclusão de curso de Patricia Paula Zardetto concluída em 2010 pela Universidade São Francisco, na área de Engenharia de Computação, as ferramentas RationalQuality Manager eRationalFunctionalTester foram utilizadas para o aumento da produtividade no desenvolvimento do software. A métrica utilizada por Zardetto, é o tempo economizado pelo o uso das ferramentas na gestão do projeto de teste de software e na execução de testes.

Pode-se observar com este estudo que a ferramenta RQM atingiu cerca de 70% das expectativas; enquanto que 20% foram atendidas mediante alguma

customização. E cerca de 10% dos testes não puderam ser automatizadospois oRQM não permite simular a máquina que executará o teste e nem simula a interface gráfica com o usuário.

A ferramenta RFT atendeu 90% das expectativas permitindo uma economia de tempo de aproximadamente 60% dos testes executados automatizados em relação à execução manual dos mesmos. Os testes automatizados correspondem a aproximadamente 25% do total de testes executados.

Este trabalho contribuiu para a verificação do aumento de produtividade um projeto de teste de software, podendo-se concluir que este aumento deve-se ao fato do uso das ferramentas de automação de gestão e de execução de testes.

A ferramenta de gestão proporcionou uma melhor organização dos documentos e das atividades e a ferramenta de execução de testes permitiu executar um maior número de casos de testes em um menor tempo assegurando a software de qualidade testado com maior rapidez e com custo reduzido.

5.3 PROCESSO DE AUTOMAÇÃO DE TESTES DE SOFTWARE COM FERRAMENTAS OPEN SOURCE: UM ESTUDO DE CASO COM INTEGRAÇÃO CONTÍNUA

Este trabalho de pesquisa criado por Cristiana YukieMasuda foi concluído em 2009 e apresentado no Campus da Grande Florianópolis da Universidade do Sulde Santa Catarina – UNISUL, para o Curso de Sistemas de Informação, com o objetivo principal garantir a qualidade do software gerando uma padronização nas atividades do processo de teste.

Utilizando para a pesquisa um produto de uma empresa de médio porte, desenvolvedora de software, do estudo de caso criou-se um projeto de testes associado ao desenvolvimento continuo do produto, sendo o processo a base para futuras confecções de projeto de testes e de desenvolvimento.

Um dos resultados do trabalho, utilizando um ambiente automatizado com apoio de algumas ferramentas open source, foi a definição do processo de teste assim como a materialização e planejamento de diversos artefatos, documentos de testes e relatórios. Os requisitos do produto, a integração dos testes e identificação de bugs,trouxe ao software maior controle de qualidade.

Expressos através denúmeros, os resultados são indicadoresimportantes que podem ser utilizados para aavaliação e tomada de decisão , de forma a minimizar as ameaças e geraroportunidades que permitam tornar a empresa cada vez mais competitiva no mercado.

6 AUTOMAÇÃO DO PROCESSO DE TESTE

Para se obter uma automação de teste bem elaborada é necessário a construção de uma metodologia com fases bem definidas, para tanto, deve-se observar as principais atividades etapas do processo de teste.

6.1METODOLOGIA

A metodologia utilizada para o desenvolvimento deste trabalho iniciou-se pela etapa de revisão bibliográfica em materiais publicados em livros, revistas, jornais, rede eletrônica, periódicos especializados, dissertações e monografias na qual foram estudados os conceitos fundamentais de engenharia e qualidade de software e a realização da automação dos testes de software.

Com a pesquisa obtida, procurou-se compreender as técnicas de testes existentes e principalmente a utilização da ferramenta automática de testes e a geração da massa de testes de acordo com a solução a ser testada.

Observou-se que os testes automatizados são programas ou *scripts* simples que exercitam funcionalidades do sistema sendo testado e fazem verificações automáticas nos efeitos colaterais obtidos. Todos os casos de teste são facilmente e rapidamente repetidos com pouco esforço, se tornando assim uma das grandes vantagens desta abordagem.

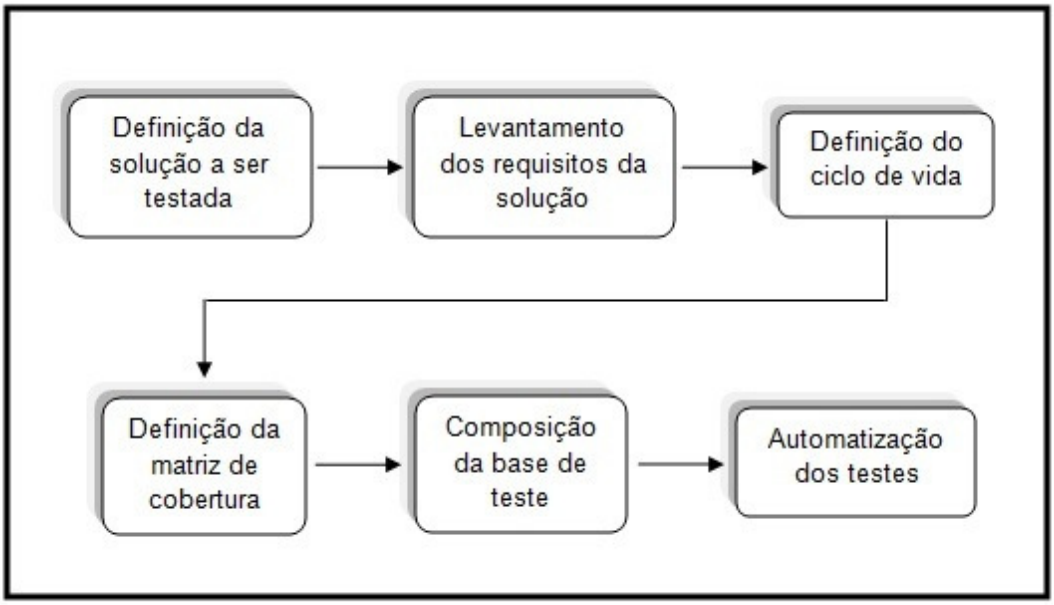
A reprodutibilidade dos testes permite executar inúmeras vezes eidenticamente situaçõesespecíficas, facilitando e encontrando possíveis comportamentos não desejados, garantindo que passos importantes não sejam ignorados por falha humana.

A confiabilidade de um software para Delamaro, Maldonado e Jino (2007) representa a qualidade do ponto de vista do usuário, onde se torna referência para as organizações a avaliação deste requisito. Para isto, o teste é extensivamente conduzido tanto para remover defeitos quanto para determinar o nível de confiabilidade.

Todas estas características ajudam a diminuir a quantidade de erros e a solucionar problemas encontrados nos testes manuais, aumentando assim a qualidade e vida útil do software.

Com base nos estudos realizados, abordamos o desenvolvimento deste trabalho implementando algumas etapas do processo de teste, conforme figura 7 abaixo:

Figura 7. Etapas do processo de teste



Fonte: Do autor

Com a sequência de teste estabelecida, abaixo serão especificados cada um dos processosenvolvidos para a criação desta metodologia.

6.2 DEFINIÇÃO DA SOLUÇÃO A SER TESTADA

A solução utilizada para o desenvolvimento dos testes automatizados foi a Nota Fiscal Eletrônica (NFS-e) do produto Flye-Nota. Utilizou-se para dar maior enfoque ao trabalho, a tela de emissão de nota fiscal.

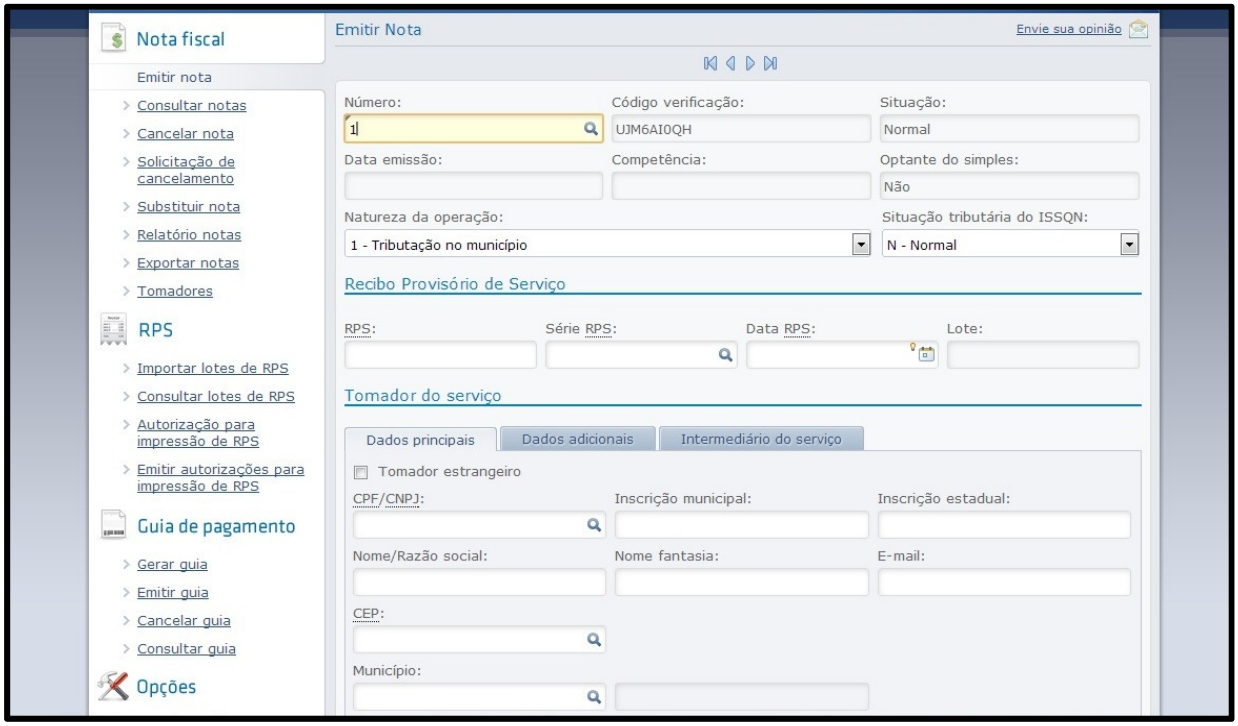
A nota fiscal eletrônica, segundo a Receita Federal é um modelo nacional de documento fiscal eletrônico que substitui a atual emissão de documento fiscal em papel, simplificando as obrigações acessórias dos contribuintes e permitindo ao mesmo tempo o acompanhamento em tempo real das operações comerciais pelo Fisco.

Esta ferramenta apresenta muitas vantagens às empresas que utilizam este sistema, como a agilidade, redução de custos com impressão, acompanhamento em tempo real da tramitação das informações, ética e

transparência na documentação de uma operação de circulação de mercadorias ou prestações de serviços.

Optou-se por esta escolha, pois a NFS-e sendo um sistema Web que possui muitos acessos e tarefas simultâneas, sendo de extrema importância que este não possua defeito algum, sendo assim sua automatização trará maior confiabilidade à empresa e aos contribuintes.

Figura 8 - Tela de emissão de nota fiscal



Fonte: Do autor

A figura acima esboça parte da tela de emissão de notas fiscais eletrônica. A mesma possui um layout claro e objetivo com campos de fácil manuseio e opções de consulta.

6.3LEVANTAMENTO DOS REQUISITOS DA SOLUÇÃO

A fase de levantamento de requisitos da solução pode ser classificada de diversas formas, tendo a finalidade de compreender a relação entre objetos, tarefas e as próprias funções do sistema.Paula Filho (2001) cita que os requisitos funcionais descrevem as funções que o produto deverá realizar em benefício dos clientes.

A forma exploratória deste trabalho foi através da classificação de requisitos funcionais da tela de emissão de nota fiscal eletrônica. Os requisitos abordados são:

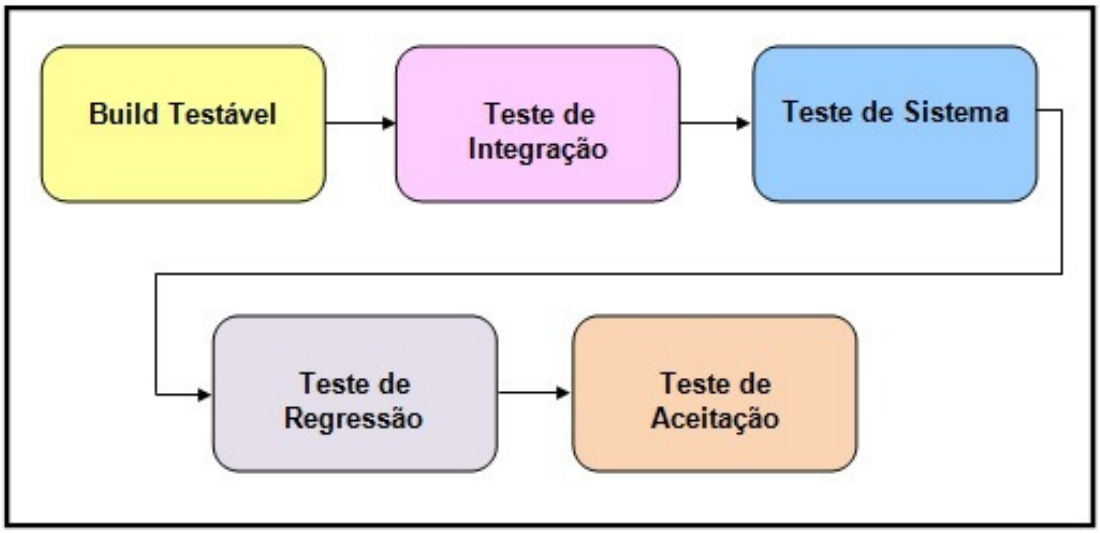
- RF01: a tela a ser testada deve deverá possuir todas as funcionalidades com acesso através do teclado e mouse;
- RF02: os campos obrigatórios devem possuir o símbolo de obrigatoriedade;
- RF03: os campos pré cadastrados devem possuir opção de consulta para facilitar;
- RF04: os campos de data devem possuir calendário disponível para consulta;
- RF04.1: o calendário possui opção para escolha do mês, ano e hora, minuto e segundo. Ainda possui o botão agora para seleção da data e hora atual.
- RF05: os campos de preenchimento automáticos devem estar desabilitados;
- RF06: o layout deve ser de claro entendimento;
- RF07: os campos devem ser habilitados ao logo do preenchimento da tela;
- RF08: os botões devem estar claramente identificados;
- RF09: as mensagens de erros e alertas devem possuir uma mensagem claro e objetiva;
- RF10: as mensagens devem ser apresentadas no campo que está sendo utilizado ou no topo da tela.
- F11: os campos de consulta rápida devem possuir uma lupa para identificação e devem ser habilitados também utilizando a tela de atalho F2.
- RF11.1: as consultas possuem um campo de busca e ordenação. Possuem também um filtro por critério e modelo.

Os requisitos citados acima foram elencados conforme o conhecimento do especialista.

6.4 DEFINIÇÃO DO CICLO DE VIDA

Um modelo de ciclo de vida organiza as macro-atividades básicas, estabelecendo precedência e dependência entre as mesmas. O ciclo de vida de testes proposto é composto por cinco fases, conforme figura 9.

Figura 9 - Ciclo de vida do teste



Fonte: Do autor

Um ciclo de vida de teste é necessário para estruturar várias atividades, suas ordens e dependências.Abaixo são especificadas cada etapa deste ciclo de vida desenvolvido.

6.4.1 Versão Testável

Após a primeira versão do software ser lançada, o teste e a análise deste não param. Segundo Pezzé e Young (2008), produtos de software em geral funcionam por muitos anos, frequentemente muito além do seu ciclo de vida previsto, sofrendo assim inúmeras mudanças. Os softwares se adaptam as mudanças no ambiente em geral, por exemplo, novos dispositivos, evolução do sistema operacional, e alteração na plataforma de banco de dados. Eles também evoluem para atender novas solicitações dos usuários e alterações em requisitos.

O build é uma versão operacional de um sistema ou parte de um sistema que demonstra um subconjunto dos recursos a serem fornecidos no produto final.

Ele é constituído de um ou mais componentes (geralmente executáveis), que são construídos a partir de outros componentes, normalmente por um processo de compilação e vinculação do código-fonte.

Tem como finalidade liberar um subconjunto testável dos recursos e funções em tempo de execução do sistema. O RationalUnifiedProcess (RUP) recomenda que uma seqüência de builds seja construída durante uma iteração, adicionando capacidade a cada um deles à medida que componentes de subsistemas de implementação forem adicionados ou aprimorados.

Quando há mudanças na implementação (novas funcionalidades, defeitos resolvidos, etc...), alterada por um desenvolvedor, uma nova build é então criada. Esta build é lançada para a equipe de testes, onde um dos colaboradores irá projetar e realizar o roteiro mais simples de testes para garantir o mínimo da cobertura de qualidade. Se uma destas solicitações de mudanças falha, o teste será reprovado e a build passa automaticamente para o codificador que adicionou anteriormente as mudanças no software.

6.4.2 Teste de Integração

Sistemas de software são formados por componentes que contribuem para a realização dos serviços prestados pelo sistema. Os testes de integração visam determinar a interoperabilidade entre esses componentes. O teste de integração é o processo de examinar se os componentes do sistema, juntos, trabalham conforme descrito nas especificações do mesmo. (PFLEEGER, 2004)

Uma vez que testados individualmente os componentes de programa, eles devem ser unidos para criar um sistema parcial ou completo. Pressman (2006) lembra que este processo envolve construir todo o sistema e testar o seu resultado quanto a problemas que surjam a partir das integrações de componentes. O desenvolvimento dos testes deve ser baseado a partir da especificação do sistema. Deve-se iniciar os testes de integração logo que as versões dos componentes do sistema estejam disponíveis.

Sommerville (2007) destaca que, localizar erros descobertos durante este processo é a principal dificuldade dos testes, pois há complexas interações entre os componentes de sistema, o que dificulta encontrar a origem do erro. Para auxiliar no processo deve-se sempre utilizar uma abordagem adicional, integrando uma

configuração mínima do sistema e testando. Em seguida inserir componentes a essa configuração mínima, que será novamente testado depois de cada novo componente adicionado.

Na fase de preparação dos testes de integração é criado um plano de integração, que deve ser executado logo após à especificação da arquitetura do sistema, afirma Pressman (2006).Esse plano deve ser revisado após a fase de implementação, pois a arquitetura do sistema pode ter sofrido alterações. Caso sejam localizados defeitos no momento da integração do sistema deve-se analisar se os problemas de integração foram originados por violação da arquitetura especificada, caso em que a implementação dever ser ajustada. Erros também podem ocorrer devido à inconsistência na arquitetura do sistema, caso em que esta deve ser revisada e corrigida. Pode ainda, ocorrer erros nos testes devido a alguma dificuldade de integração, como à má configuração do ambiente.

6.4.3 Teste de Sistema

Pezzé e Young (2008) consideram o teste de sistema o ponto culminante do teste de integração. Estes testes verificam se o sistema, composto pela integração de componentes desenvolvidos (novos) e de terceiros, atende aos requisitos funcionais e aos atributos de qualidade desejados, tais como desempenho, usabilidade, portabilidade e segurança.

O teste de sistema é considerado por Bastos et al. (2007),Pressman (2006), Inthurn (2001), Braude (2005), Rios e Moreira (2006), como a execução de um sistema como um todo, e este deve executar o que lhe foi proposto.

Rios e Moreira (2006, p. 14) lembram que “[...] neste estágio de testes deve ser simulada a operação normal do sistema, sendo testadas todas as suas funções de forma mais próxima possível do que irá ocorrer no ambiente de produção”.

A independência do teste de sistema impede a repetição dos erros de projeto do software no projeto de teste, salienta Pezzé e Young (2008). Esse risco existe em alguma proporção em todos os estágios do desenvolvimento, mas sempre na permuta por algum benefício ao projetar casos de teste eficientes baseados na similaridade com o projeto de software e suas ciladas potenciais.

Algumas propriedades do sistema são essencialmente globais, incluindo propriedades de desempenho e propriedades de confiança, como tempo médio entre falhas. A importância de tais propriedades gerais é ampliada durante o teste de sistema.

Os principais objetivos dessa técnica mencionam Rocha, Maldonado e Weber (2001), Inthurn (2001) é descobrir erros de funções e implementações que não estão em conformidade com os requisitos especificados. Simplificando, se trata de encontrar os defeitos graves antes dos clientes.

6.4.4 Teste de Regressão

O teste de regressão é realizado em todo sistema ou em parte dele, depois que alguma mudança foi realizada na aplicação, esclarecem Bartié (2002), Bastos et al. (2007), Delamaro, Maldonado e Jino (2007), Molinari (2003), Pezzé e Young (2008), Rios e Moreira (2006). Seu objetivo principal é verificar se as funcionalidades existentes foram afetadas com as mudanças.Além deste Bastos et al. (2007) cita mais alguns:

- Verificar se a documentação do sistema permanece atualizada;
- Verificar se os dados e as condições de teste permanecem atuais.

Uma abordagem simples de teste de regressão, de acordo com Pezzé e Young (2008), consiste em re-executar todos os casos de teste elaborados para as versões anteriores. Uma suíte de teste de boa qualidade pode ser mantida por varias versões do sistema, removendo casos obsoletos e marcando casos redundantes.

O grande problema para execução desse teste, aponta Bastos et al. (2007), Pezzé e Young (2008), são os gastos excessivos de tempo e de dinheiro. Dependendo da característica e necessidade de cada sistema, uma ou mais técnicas de testes pode ser utilizadas.

A documentação dos testes se faz muito importante para os testes. As especificações de teste definem as funcionalidades a serem testadas, os casos de teste apropriados, as entradas e os resultados esperados, bem como as condições de execução para todos os casos existentes. Os relatórios indicam os resultados da execução dos testes, as falhas encontradas e suas ligações com os casos de teste.Os resultados desses testes ajudam a avaliar a qualidade do produto.

6.4.5 Teste de Aceitação

A técnica do teste de aceitação é executada pelo cliente com o objetivo de verificar se o sistema desenvolvido atende aos seus objetivos de negócio e a seus requisitos, em termos de funcionalidade e usabilidade, explicam Molinari (2003), Bastos et al. (2007), Pfleeger (2004), Rios e Moreira (2006).

Uma versão do sistema pode ser liberada para os usuários, que darão retorno sobre as falhas e a usabilidade. Este teste é chamado de alfa e beta, e distinguem etapas de teste, explica Pezzé e Young (2008). Usualmente as fases preliminares ou conhecidas ainda como alfa são executadas na organização de desenvolvimento, enquanto que as últimas ou também chamadas de beta são executadas nos *sites* dos usuários.Bartié (2002) divide esta fase em três momentos: o aceite formal, alfa teste e beta teste, conforme figura 10.

Tabela 3. Estágios do processo de aceite de um software

Homologação (Aceite da Solução)		
Aceite Formal	**Implantação Alpha**	**Implantação Beta**
Clientes planejam e realizam os testes do software.	Clientes são convidados a operar o software no fornecedor.	Clientes selecionados recebem o software para operar em seu ambientes.

Fonte: Adaptado de Bartié (2002, p. 158)

Inthurn (2001) fala que a validação é bem sucedida quando o software funciona da maneira esperada pelo cliente ou conformeespecificado na fase de projeto.

Algumas vantagens são apontadas por Bastos et al. (2007) e Pfleeger (2004) em relação a este teste:

- Detecção antecipada de possíveis erros, possibilitando também, que novos prazos venham a ser negociados;
- Mais eficiência na execução dos testes, na preparação do plano e nos casos de teste devido à participação dos usuários;
- Permitem determinar novas necessidades e novos requisitos pelo usuário.

Bastos et al. (2007) ainda lembra que esse tipo de teste não precisa necessariamente ser realizado no final do desenvolvimento, pode ser realizado

durante todas as suas etapas, assim que um produto intermediário estiver pronto para o teste.

6.5 DEFINIÇÃO DA MATRIZ DE COBERTURA

A matriz de rastreabilidade, também conhecida como *Traceability Rastreability Matrix (TRM)*, se trata de um documento de acompanhamento que facilita a visualizacao dos relacionamentos entre requisitos e outros artefatos ou objetos. A rastreabilidade entre requisitos e casos de teste permite, entre outras coisas, visualizar quantos casos de testes foram alocados para um requisitos e se algum dos mesmos não possui casos de testes alocados.

O cubo mágico, como também é conhecida a matriz de rastreabilidade, foi construindo utilizando o Microsoft Excel e o mesmo é constituído de uma planilha principal que possui os requisitos não funcionais dos testes, uma segunda com o planejamento funcional e outras oito planilhas que possuem os casos de testes que são relacionados a planilha principal.

- A planilha principal é formada por requisitos não funcionais do sistema, sendo estes, restrições e comportamentos que o software deve satisfazer, e que compõem as três baterias de testes criadas. Para garantir a cobertura de testes deste trabalho foi utilizada a bateria 01.

Os requisitos escolhidos para formar cada bateria foram: a versão do Java utilizada na estação de trabalho, os browses utilizados para a execução dos testes, o sistema operacional instalado e se utilizada máquina virtual. Este primeiro documento do cubo mágico possui um link para acesso direto às planilhas de casos de teste, como é mostrado na coluna ciclo da figura 10. É apartir desta planilha que será possível verificar os casos de testes planejados e executados pela ferramenta de automação.

Figura 10 – Bateria 01 da aba principal da matriz de rastreabilidade.

Bateria 01				
Ciclo	Versão do Java	Browsers	Sistema Operacional	Máquina Virtual
TRM 01	6.0	Internet Explorer	Windows	Nenhuma
TRM 02	6.0	Internet Explorer	Windows	Nenhuma
TRM 03	6.0	Internet Explorer	Windows	Nenhuma
TRM 04	6.0	Internet Explorer	Windows	Nenhuma
TRM 05	7.0	Internet Explorer	Windows	Nenhuma
TRM 06	7.0	Internet Explorer	Windows	Nenhuma
TRM 07	7.0	Internet Explorer	Windows	Nenhuma
TRM 08	7.0	Internet Explorer	Windows	Nenhuma

Fonte: Do autor

- O planejamento funcional está contido na segunda planilha da matriz de rastreamento. Este documento abrange os testes de integração, testes de sistema e teste de regressão, que foram estruturados para serem cumpridos dentro de um cronograma de três semanas, elaborados com o conhecimendo especialista.Definido os requisitos para a validação em conjunto com as baterias de testes, o mais importante dentre eles é a emissão de nota fiscal eletrônica, que será o alvo da composição deste trabalho.

Figura 11 – Planejamento funcional da matriz de rastreabilidade.

Teste de Integração			
Planejamento			
Semana 1	Semana 2	Semana 3	Requisitos para validação
Bateria 01	Bateria 01	Bateria 01	Emissão de nota fiscal eletrônica
Bateria 02	Bateria 02	Bateria 02	Consultar notas
Bateria 03	Bateria 03	Bateria 03	Cancelar notas
Teste de Sistema			
Planejamento			
Semana 1	Semana 2		Requisitos para validação
Bateria 01	Bateria 01		Emissão de nota fiscal eletrônica
Bateria 02	Bateria 03		Consultar notas
Bateria 03	Bateria 02		Cancelar notas
Teste de Regressão			
Planejamento			
Semana 1			Requisitos para validação
Bateria 01			Emissão de nota fiscal eletrônica
Bateria 02 Bateria 03			Consultar notas
Bateria 02 Bateria 03			Cancelar notas

Fonte: Do autor

Para os testes de integração as baterias foram incorporadas às três semanas do planejamento, pois é nesta fase em que todos os componentes são unidos e que os testes devem cobrir todas as áreas integradas, dispendendo maior tempo para a execução dos testes.

Utilizando duas semanas do planejamento para a execução das baterias, o teste de sistema foi projetado para executar os testes em uma ordem que atenda a demanda da validação dos requisitos, verificando se estão de acordo com o que foi solicitado.

O teste de regressão será realizado em uma semana e as baterias de testes necessárias para suprir a qualidade foram incorporadas a ela. Este teste se torna menor pois o seu objetivo é averiguar se as funcionalidades já existentes foram afetadas com as mudanças realizadas no software.

- As TRM criadas são diretamente ligadas a planillha principal da matriz de rastreabilidade. Nelas estão todos os casos de testes descritos de um conjunto de ações ou valores de entrada. Para cada bateria de teste formulada, os casos de testes contidos em uma TRM podem ser planejados e executados. Caso os testes tenham sido executados na ferramenta de automação *TestComplete*e realizados com sucesso a coluna “Passou” da planilha deve ser marcada juntamente com o usuário que testou na coluna “Quem testou”. Mas se o caso de teste não obtiver sucesso e encontrar uma falha a mesma deve ser marcada na coluna falhou, o erro deve ser salvo em uma ferramenta que reporte as falhas e o seu código deve ser marcado na coluna “Cód. erro”, juntamente com a “Data” e “Quem testou”.

Figura 12 – TRM de casos de teste.

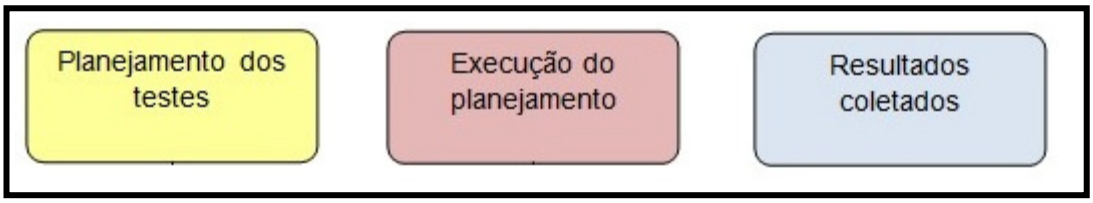
Cod#	Caso de teste	Planejado	Executado	Passou	Falhou	Cód. Erro	Data	Quem testou
277	Preencher o campo Nome / Aba Intermediário do serviço com números negativos e letras.							
278	Preencher o campo Nome / Aba Intermediário do serviço com letras.	x	x	x				
279	Preencher o campo Nome / Aba Intermediário do serviço com letras e caracteres especiais.							
280	Preencher o campo Nome / Aba Intermediário do serviço com caracteres especiais.							
281	Preencher o campo Nome / Aba Intermediário do serviço com vazio.							
282	Preencher o campo Serviço com números aleatórios.							
283	Preencher o campo Serviço com números negativos aleatórios.	x	x	x				
284	Preencher o campo Serviço com vazio.	x	x	x				
285	Preencher o campo Serviço com os dados pré cadastrados no sistema.	x	x	x				
286	Preencher o campo Serviço utilizando a tecla de atalho F2.	x	x	x				
287	Preenchimento automático do campo Alíquota de acordo com a seleção anterior.	x	x	x				
288	Preencher o campo Alíquota com zero.	x	x		x	79432	30/10/2012	Tester 2
289	Preenchimento do campo Prestado no país com "Sim" ou "Não".							
290	Preencher o campo Município com números aleatórios.							
291	Preencher o campo Município com números negativos.							
292	Preencher o campo Município com números e caracteres especiais.							
293	Preencher o campo Município com números e letras.							
294	Preencher o campo Município com números negativos e caracteres especiais.							
295	Preencher o campo Município com números negativos e letras.							
296	Preencher o campo Município com letras.	x	x	x				
297	Preencher o campo Município com letras e caracteres especiais.							
298	Preencher o campo Município com caracteres especiais.							
299	Preencher o campo Município com vazio.							
300	Preencher o campo Município com a sequência de números pré configurados no sistema.	x	x	x				
301	Não preenchimento do campo Município quando a seleção anterior for igual a "Não".							
302	Preencher o campo Município utilizando a tecla de atalho F2.							

Fonte: Do autor

Na figura acima é apresentada a TRM 01, utilizada para especificar os casos de teste planejados e executados na emissão de nota fiscal eletrônica. Pode-se observar que um erro foi reportadopela ferramente de automação e sua erro foi devidamente cadastrada.

Com base na matriz de rastreamento criada, são executadas as seguintes tarefas apontadas na figura 13.

Figura 13 – Planejamento dos testes



Font
e: Do autor

- O planejamento deu-se inicialmente pela identificação do item a sertestado e pela definição dos pontos principais e fundamentais desta solução. Em seguida foram estudadas as maneiras de como cada parte integrante do software a ser testado deve ser submetido ao teste, quais as informações de entradas necessárias para o fornecimento das saídas corretas e a maneira com que o este software deve se comportar ao passar por uma série de testes.

Logo após foram elaborados os casos de testes para cada campo válido da solução, criando assim casos com valores de entrada e saídas esperados para cada instância de teste. A especificação dos casos se deram em ordem de execução da automatização, pois a execução correta de um caso de teste pode depender de um estado de uma base de dados, que é produzido por um caso executado anteriormente.

A criação da matriz de rastreabilidade, ou cubo mágico, foi realizada na seqüência, onde houve a ligação dos critérios não funcionais do ambiente de teste, juntamente com o planejamento dos requisitos de validação e suas baterias de testes e a composição dos casos de testes.

Por fim foi realizada a implementação dos testes, onde a ferramenta e o ambiente de teste para a automatização foram totalmente preparados, tornando-se disponíveis todos os recursos necessários. Podem resultar, dos problemas encontrados nesta execução, solicitação de correção de itens, alterações nos casos de testes entre outros.

- Os casos de testes agrupados na matriz de rastreabilidade na aba TRM 01 foram planejados, de acordo com o conhecimento do especialista, em cruzamento com a bateria de teste 01 da aba principal. Estes casos planejados foram automatizados na ferramenta *TestComplete*. Após a execução de cada um dos casos planejados a matriz foi atualizada com os dados de acertos e erros.
- Com os resultados coletados com o desenvolvimento deste trabalho, pode-se chegar ao seu objetivo principal, que é elaborar uma solução automatizada para testes de software de acordo com o que é proposto pelas normas e técnicas especificadas através da metodologia criada.

A validação da metodologia desenvolvida, proporcionou uma base sólida de informações que serviram como apoio a decisão para o analista de testes. A cobertura funcional dividida em ciclos de integração (3 semanas), ciclo de sistema (2 semanas) e ciclo regressão (1 semana), garantiram cobertura funcional e cobertura não funcional de 100% para o escopo da solução testada.

A matriz implementada, considerou a experiência do especialista como fator de derivação funcional, gerando 390 casos de testes para cada um dos requisitos de validação. Alem disso, com a inclusão e validação das variáveis não funcionais e com seu cruzamento com as demais informações, a configuração se tornou válida.

A combinação da metodologia adotada com a técnica de execução de testes automatizados com a ferramenta *TestComplete*,fundamentou e complementou a agilidade dos testes bem como o aumento da qualidade, uma vez que a fase de regressão obteve maior cobertura, equiparando-se a fase de integração.

Apesar da cobertura funcional ter abordados poucos requisitos, pode-se comprovar que o planejamento e a execução baseado na metodologia desenvolvida direcionaram os testes, fazendo com que o testador garantisse a qualidade nos pontos mais relevantes, e conseguisse assumir o risco de não executar testes utilizando todas as baterias e todos os casos de testes, garantindo qualidade e aumentando o custo benefício da fase de teste

Foram executados 84 testes na ferramenta de automação conforme o planejamento especificado na matriz de rastreamento. Destes casos de testes automatizados 83houveram sucesso em sua execução não encontrando erros no software, e apenas um deles falhou. Assim que encontramos um *bug* o reportamos através de uma ferramenta de *trancking* para que em seguida o mesmo seja

concertado. Então é cadastrada a *erro* com a falha encontrada, que gerará uma numeração que será armazenada na matriz de rastreamento.

Com a criação deste trabalho também foi possível a elaboração de dois artigos que foram exposto na seção oral do VI SULCOMP. Destes artigos que foram elaborados por mim e pela colega de curso Carina Cardoso, um é referenciado a este projeto, e o outro é referenciado ao projeto de conclusão de curso da Carina que leva o nome de “Utilização do EPF Composer para o desenvolvimento de um metamodelo baseado no MPS.BR para a criação de processos aplicados à garantia da qualidade em projetos de software.”

6.6 COMPOSIÇÃO DA BASE DE TESTE

Para Delamaro, Maldonado e Jino (2007) testar todos os valores de entrada ou executar todos os caminhos em um programa é idealmente desejável, mas impraticável. Se torna essencial para que o projeto seja bem sucedido, selecionar os requisitos mais apropriados e importantes para o teste. Os escolhidos para verificação representarão um equilíbrio entre o custo, o risco e a necessidade de verificá-los.

A identificação dos casos de teste é importante por vários motivos. Abaixo são listados alguns deles:

- Os casos de teste constituem a base do design e do desenvolvimento dos *Scripts* de Teste.
- A "profundidade" do teste é proporcional ao número de casos de teste. O aumento do número de casos de teste gera uma maior confiança na qualidade do produto e no processo de teste, já que cada caso de teste reflete um cenário, uma condição ou um fluxo diferente através do produto.
- A principal avaliação da abrangência do teste é a cobertura baseada em requisitos, de acordo com o número de casos de teste identificados, implementados e/ou executados. Uma sentença como "Executamos e verificamos 95% dos casos de teste críticos" é mais significativa do que a sentença "Já executamos 95% do total de testes".

- A escala do esforço de teste é proporcional ao número de casos de teste. Com uma análise abrangente dos casos de teste, é possível estimar com mais precisão a duração dos estágios subseqüentes do ciclo de teste.
- Os tipos de design e desenvolvimento de testes e os recursos necessários são amplamente controlados pelos casos de teste.

A técnica utilizada neste trabalho para a composição da base de teste foi a de gerar dados conforme o conhecimento do especialista.Utilizando a técnica de derivação funcional, que une especificações de requisitos, análise e projeto, os casos de teste foram conduzidos na interface do software, conforme figura 14. Esta técnica, conforme Rocha, Maldonado e Weber (2001), são empregados para demonstrar que as funções do software estão operando corretamente, que a entrada é adequadamente aceita e a saída é corretamente produzida e que a integridade da informação externa (uma base de dados, por exemplo) é conservada.

Figura 14 – Base de testes

ID	Caso de teste	Resultado esperado
Campo Tomador estrangeiro:		
TC 155	Não preencher o campo com check habilita os campos abaixo.	Válido
Campo CPF/CNPJ:		
TC 156	Preencher o campo com números aleatórios.	Mensagem de erro
TC 157	Preencher o campo com números negativos.	Mensagem de erro
TC 158	Preencher o campo com números e caracteres especiais.	Mensagem de erro
TC 159	Preencher o campo com caracteres especiais.	Mensagem de erro
TC 160	Preencher o campo com vazio.	Mensagem de erro
TC 161	Preencher o campo com números nos formatos xxx.xxx.xxx-xx ou xx.xxx.xxx/xxxx-xx.	Válido
Campo Inscrição municipal:		
TC 162	Preencher o campo com números.	Válido
TC 163	Preencher o campo com números negativos.	Válido
TC 164	Preencher o campo com números e caracteres especiais.	Válido
TC 165	Preencher o campo com números e letras.	Válido
TC 166	Preencher o campo com números negativos e caracteres especiais.	Válido
1C 167	Preencher o campo com números negativos e letras.	Válido
TC 168	Preencher o campo com letras.	Válido
TC 169	Preencher o campo com letras e caracteres especiais.	Válido
TC 170	Preencher o campo com caracteres especiais.	Válido
TC 171	Preencher o campo com vazio.	Válido

Fonte: Do autor

A organização da base de teste foi projetada envolvendo várias possibilidades das funções do software, onde se fez necessário elaborar pequenas situações para cada componente da tela, para que os mesmos fossem testados individualmente. Deste modo, criou-se situações válidas e inválidas.

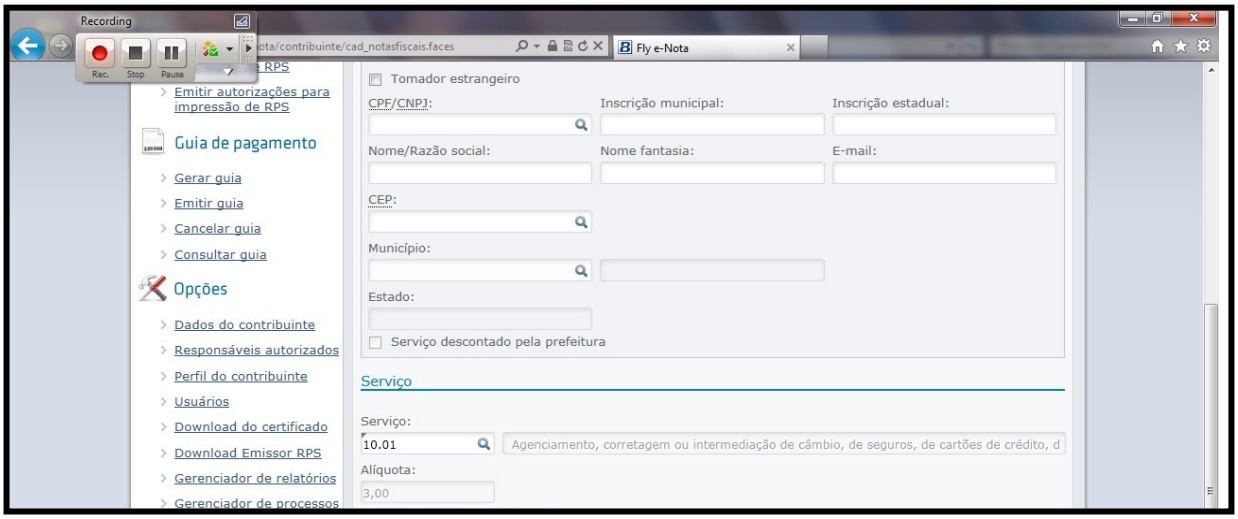
6.7 AUTOMATIZAÇÃO DOS TESTES

Planejar a descrição precisa do comportamento esperando de um teste, por diversas vezes se torna algo muito difícil. Conforme Pezzé e Young (2008) quando muitas características de um programa com interface gráfica podem ser elencadas de uma forma adequada para auto-teste, algumas propriedades requerem uma pessoa para interagir com o programa e julgar seu comportamento. Não sendo possível evitar completamente o envolvimento humano na execução dos testes, pode-se pelo menos evitar a repetição desnecessária.

Para a automação dos testes da emissão de notas fiscais, foi utilizada a técnica de programação de *scripts* Captura e Reprodução *(Capture-Replay).* Este método grava (*Capture)* as ações do usuário interagindo com a aplicação, gerando um *script* de teste que pode ser reproduzido *(Replay).*

O processo de automação será executado conforme os casos de testes planejados na matriz de rastreabilidade e o primeiro passo para esta implementação foi a criação de um projeto no programa TestComplete utilizando a linguagem Java, pois a solução a ser testada é desenvolvida nesta linguagem. Dentro deste projeto, deve-se acionar o botão de gravação de *scripts*, que em seguida já gravará todas as ações executadas. O browse do Internet Explorar é então aberto e o login para o acesso à página de emissão de notas é feito, conforme figura 15. Na sequencia um dos casos de testes planejados no cubo mágico é gravado e após esta ação o botão *Stop*da caixa de diálago *Recording*é executado.

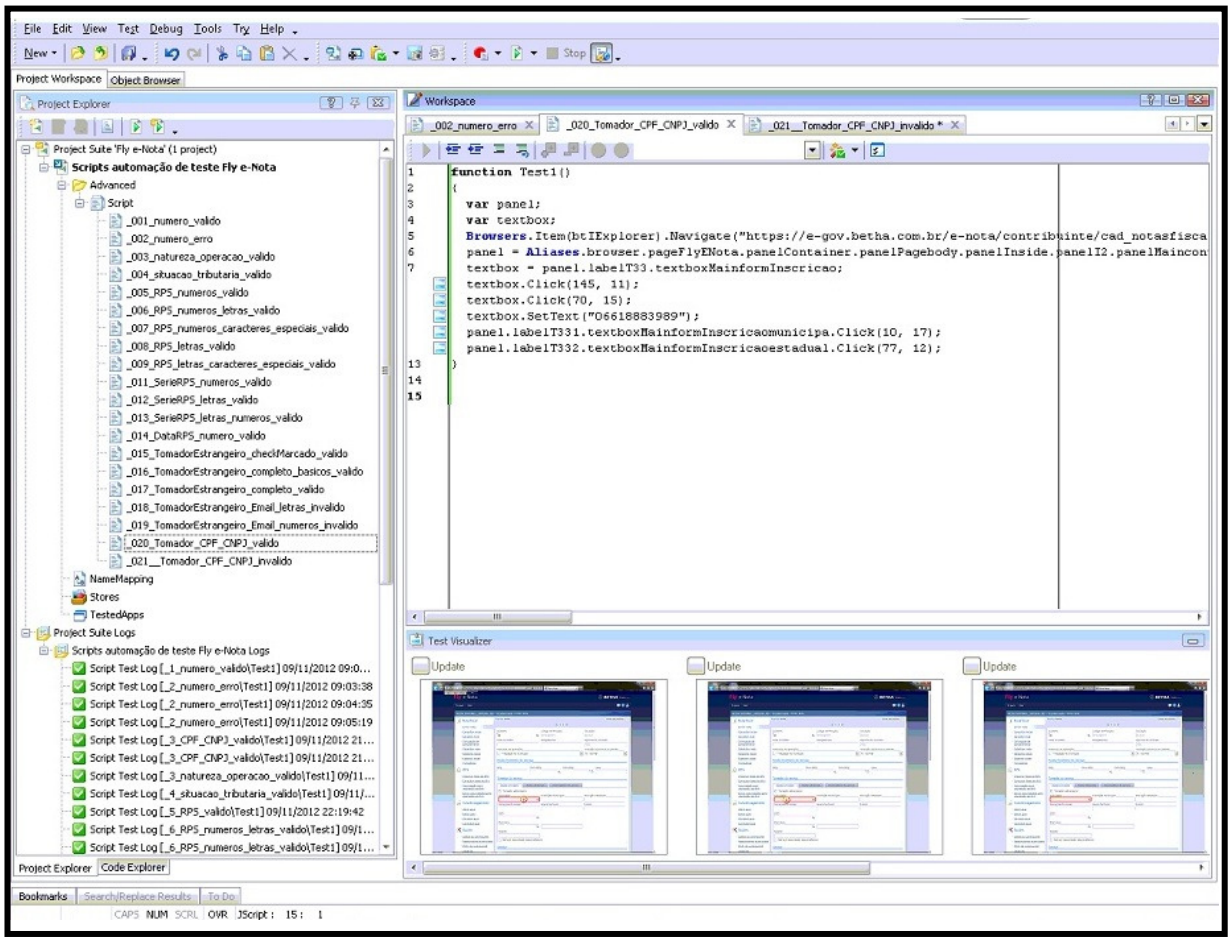
Figura 15 – Gravando caso de teste no TestComplete.



Fonte: Do autor

Após a gravação de um caso de teste, a ferramenta cria um novo procedimento e converte todas as instruções realizadas na linguagem do *script* alvo, neste caso em Java, como pode ser visto na figura 16. Todas as ações são transformadas em instruções de fácil entendimento, contribuindo assim para a criação e manutenção dos *scripts.*

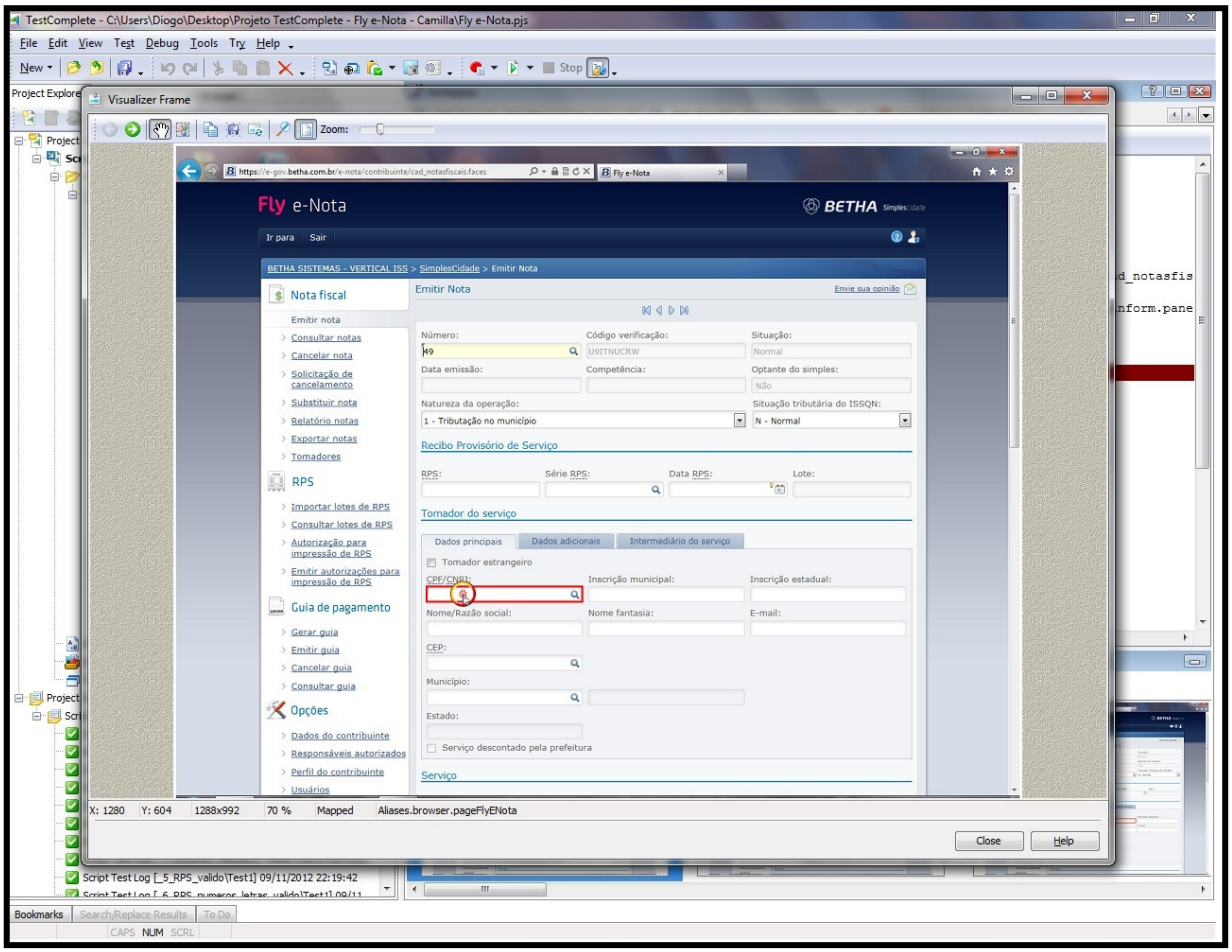
Figura 16 – Script gravados no TestComplete.



Fonte: Do autor

O *TestComplete* oferece um completo ambiente de desenvolvimento para a criação e manutenção dos scripts. Possui também recursos para depuração (*debug*) dos scripts de testes e é possível visualizar os passos realizados no momento da gravação dos *scripts*, conforme figura 17.

Figura 17 – Visualização da gravação de um caso de uso no TestComplete.

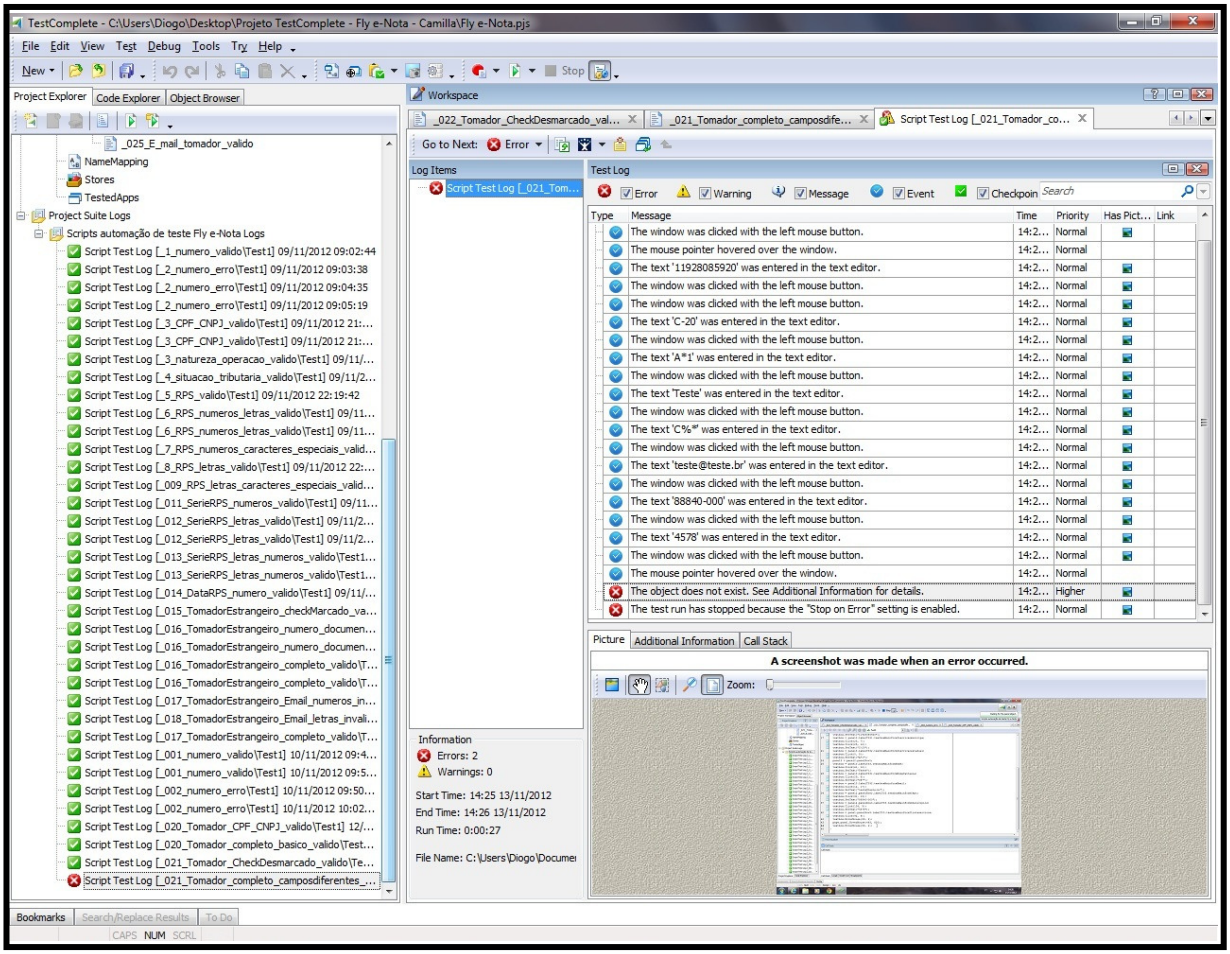


Fonte: Do autor

Após um script ser gravado, o mesmo pode ser executado outras vezes onde a ferramenta executará todas as ações da mesma forma e ordem em que foram gravadas. Por este motivo há a necessidade de as gravações serem em ordem, para que as execuções não falhem.

Ao término das gravações o programa exibe um relatório de execução que apresenta algumas informações sobre a execução dos testes, erros ocorridos, ações realizadas entre outras, que é chamada de *Test Log,* conforme figura 18.

Figura 18 – Log das execuções.



Fonte: Do autor

7 CONCLUSÃO

A necessidade de criação de softwares com grande qualidade aumentou a importância dasatividades de teste de software. Com o avançotecnológicoas técnicas de teste de software puderam evoluir, dentre elas a automação dos testes.

Através do desenvolvimento da metodologia com a criação do ciclo de vida do software, juntamente com a matriz de rastreabilidade que fez oscruzamentos das baterias de testes dos requisitos não funcionais com os requisitos de validações e os casos de testes criados, juntamente com a automatização dos testes utilizando uma solução *WEB* foi possívelconfirmar o planejamento da proposta deste trabalho.

Através de ferramentas especializadas, a automatização surgiu com o intuito de beneficiar a produtividade dos projetos de desenvolvimentos. Existem várias ferramentas disponíveis. Dentre elas, as ferramentas de gestão e deexecução de testes, foco de estudo deste trabalho.

Muitos dos casos de testes ainda requerem execução manual, porém os testes que podem ser automatizados apresentam considerável ganho de produtividade principalmente em relação ao tempo de execução. Com a realização deste trabalho, pode-se demonstrar que novas soluções podem contribuir e agilizar a fase de testes de software, considerando que é uma das fases mais importantes no ciclo de desenvolvimento quando se fala em qualidade.

Não se tem dúvida de que a automatização de testes é uma solução que alcança a agilidade necessária, porém também demanda custos, e tempo para a criação e manutenção dos scripts. Por este motivo, muitas vezes esta solução não é adotada pelas empresas de software. Através da ferramenta desenvolvida buscou-se diminuir o tempo de implementação destes scripts, gerando automaticamente os testes, para que posteriormentesejam executados por ferramentas adequadas, no caso a ferramenta *TestComplete*.

7.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

A partir do trabalho apresentado, as recomendações para trabalhos futuros são as seguintes:

- Validar a solução em maior abrangência;
- Aplicar o trabalho em uma organização;

- Validar uma maior quantidade de requisitos funcionais e não funcionais;
- Ajustar o tempo de execução dos casos de teste.

REFERÊNCIAS

Artigo Engenharia de Software 3 - A Importância dos Testes Automatizados Metodologias Ágeis - A Importância dos Testes Automatizados. **Revista Engenharia de Software Edição 3.**

AUTOMATEDQA CORPORATION. GettingstartedwithTestComplete 9. [S.l.], [2012]. Disponível em: <http://downloads.smartbear.com/Docs/Getting_Started_With_TestComplete.pdf>. Acesso em: 02 outubro de 2012.

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

BARTIÉ, Alexandre. **Garantia da Qualidade de Software.** Rio de Janeiro: Elsevier, 2002.

BASTOS, Anderson et al. **Base de conhecimento em teste de software.** 2 ed. São Paulo: Martins, 2007.

BEN-ARI, M.**The Bug that Destroyed a Rocket.Journal of Computer Science Education,** v. 13, n. 2, p. 15—16, 1999.

BRAUDE, Eric. **Projeto de Software.** Porto Alegre: Bookman, 2005.

CAETANO, Cristiano. **Introdução à Automação de Testes Funcionais.** Disponível em:< http://www.testexpert.com.br/?q=node/178> . Acesso em: 26/06/2012.

CAETANO, Cristiano. **Automação e Gerenciamento de Testes: Aumentando a Produtividade com as Principais Soluções Open Source e Gratuitas**. Disponível em:<http://www.testexpert.com.br/?q=node/795>. Acesso em: 26/06/2012.

COSTA, Mozart Guerra. **Estratégia de automação em testes: requisitos, arquitetura eacompanhamento de sua implantação**. 2006. 93 f. Dissertação (Mestrado) - Instituto deComputação, Universidade Estadual de Campinas, Campinas, 2006.

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

FELIPE, Pabro Fernando. C.M.M.I. e Spice: **Um estudo comparativo na abordagem da engenharia de requisitos**. <www.cin.ufpe.br/~pmr/qualidade/Monografia.pdf>. Acessado em: 14 de junho 2012 às 21:30.

MYERS, G. J. et al**.** The Art of Software Testing.[S.l.]:Wiley, 2004.

G. J. Myers, C, Sandler, T. Badgett e T. M. Thomas. *The Art of Software Testing*.Jhon Wiley & Sons, Nova York, NY, EUA, 2.ed., 2004.

INTHURN, Cândida. Qualidade & teste de software. Florianópolis: VisualBooks, 2001.

MOLINARI, Leonardo. **Testes de software**: produzindo sistemas melhores e mais confiáveis. 2 ed. São Paulo: Érica, 2003.

MOLINARI, Leonardo. **Inovação e Automação de testes de software**. São Paulo: Editora Érica Ltda, 2010. 140 p.

MOREIRA FILHO, Trayahú R.; RIOS, Emerson. Projeto & engenharia de software: teste de software. Rio de Janeiro: Alta Books, 2003.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software**: fundamento, métodos e 140 padrões. 2 ed. Rio de Janeiro: LTC, 2001.

PFLEEGER, Shari Lawrence. **Engenharia de Software**: teoria e prática. 2 ed. São Paulo: Prentice Hall, 2004.

PRESSMAN, R. S. **Engenharia de Software.** 6. ed. Rio de Janeiro: McGraw-Hill,2006, 720p.

PEZZÉ, Mauro; YOUNG, Michael. **Teste e Análise de Software**: processos, princípios e técnicas. Porto Alegre, 2008.

REZENDE, Alcides Denis. **Engenharia de Software e Sistemas de Informação**. 3 ed. Rio de Janeiro: Brasport, 2005.

RINCON, André Mesquita. **Qualidade de Software**. Disponível em: <http://www.clebertoledo.com.br/blogs/tecnologia/administracao/files/files/Qualidade_de_Software.pdf>. Acessado em: 14 de junho 2012 às 21:19.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software.** 2 ed. Rio de Janeiro: Alta Books, 2006.

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

SOFTEX, 2009 ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DOSOFTWARE BRASILEIRO – SOFTEX. MPS.BR – Guia Geral:2009. Disponível em: www.softex.br.

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

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

APÊNDICE A – ARTIGO CIENTÍFICO

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

Camilla Damiani[1], Gustavo Bisognin[2]

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

[2]MSc. Professor do Curso Ciência da Computação - Universidade do Extremo Sul Catarinense (UNESC) - Criciúma – Brasil

camilla_damiani@hotmail.com, gbisog@gmail.com

Abstract. *Withthegrowingdemand for computer systems, it isobservedthatthedemand for software thathave a provenlevelofqualityisincreasing. The testingactivity plays a key role in ensuringthatqualityandidentifydefectsearly in the software, thusreducingtheworkofthedevelopersfixthem later whenthe software isnowshippingtocustomers. Lifecycleswellprepared are neededtostructurevariousactivities, theirordersanddependencies. Test automationaimstoreducelong-termcostsoftesting, speedandefficiency. . Withautomation, thetestsbecome more accurateand are rapidlyfailures.*

Resumo. *Com o crescimento da demanda por sistemas computacionais, observa-se que a exigência por software que apresentam um grau comprovado de qualidade, é cada vez maior. A atividade de teste tem papel fundamental em certificar essa qualidade e identificar defeitos no software precocemente, diminuindo assim o trabalho dos desenvolvedores em corrigi-los posteriormente quando o software já está sendo distribuído aos clientes. Ciclos de vida bem elaborados são necessários para estruturar várias atividades, suas ordens e dependências. A automação de teste visa em longo prazo à redução dos custos com testes, a rapidez e eficiência. . Com a automatização, os testes tornam-se mais precisos e encontram-se rapidamente as falhas.*

1. Introdução

O mercado atual apresenta uma grande quantidade de sistemas computacionais com baixa qualidade. A realidade das organizações desenvolvedoras de sistemas computacionais é bastante crítica, apresentando projetos com escopo complexo e prazos curtos, o que fomenta o desenvolvimento de software sob pressão e aumenta consideravelmente a probabilidade de inserção de defeitos no processo de desenvolvimento.

Outro fator bastante crítico referente à qualidade dos sistemas desenvolvidos atualmente, é a falta de cobertura de testes, fazendo com que a responsabilidade sob a qualidade da solução fique totalmente com o desenvolvedor, que muitas vezes, abdica da qualidade em detrimento do prazo de entrega.

Com o crescimento da demanda por sistemas computacionais, observa-se que a exigência por software que apresentam um grau comprovado de qualidade, é cada vez maior. Neste contexto, entra a Engenharia de Software com os modelos de qualidade que possuem o objetivo de apoiar o processo de desenvolvimento a fim de garantir a qualidade, desde a fase de concepção até a liberação para a produção.

Uma das formas de obter a complementação perfeita entre o conhecimento multidisciplinar e as técnicas de engenharia de software existentes no mercado, é a utilização da experiência do analista de negócio aplicada à composição da massa de testes no processo de derivação funcional.

Considerando todos os pontos apresentados anteriormente e os resultados obtidos após a realização do trabalho, sendo que a realização da automação do processo de teste de software foi motivada como estratégia para garantir a qualidade dos sistemas desenvolvidos.

2. Teste de Software

Atualmente a usabilidade de softwares mostra-se fundamental no mercado empresarial, o que faz com que profissionais da área de Tecnologia da Informação (TI), se preocupem com erros existentes em seu produto, pois em uma organização inúmeras atividades dependem do seu correto funcionamento. Para tanto o presente capítulo aborda o teste de software, tipo de testes de software, técnicas e métodos de teste.

Processos de desenvolvimento e padrões aplicados a um software estão diretamente relacionados a sua qualidade. O teste de software é uma importante etapa deste processo, cuja função é encontrar erros, certificando assim, a qualidade do programa testado.

Conforme Rios e Moreira (2006), nas décadas de 1960 e 1970 no processo de desenvolvimento a maior preocupação era em relação a codificação e não os testes. Nessa época os testes eram feitos pelos desenvolvedores e se comparam hoje com os testes unitários e de integração. Uma maior importânciapara os testes passou-se a se dar a partir da década de 80 quando este começou a ser tratado como processo formal do desenvolvimento.

Pezzè e Young (2008) definem o teste de software como uma atividade que tem por objetivo validar e verificar o produto, revelando defeitos e aumentando a sua confiabilidade. Molinari (2003), completa dizendo que além de encontrar erros o mais cedo possível, a função do teste é fazer com que eles sejam ajustados.

Paula Filho (2001) diz que defeitos são encontrados primeiro nas revisões onde são mais eficazes de encontrá-los e corrigi-los. A fase de teste é muito importante, pois alguns erros que podem passar desapercebidos pelas revisões, serão encontrados então nesta fase, garantindo a confiabilidade do software.

As práticas de teste, para Pezzè e Young (2008), começam antes mesmo que o código esteja finalizado. Para que erros surgidos nas fases iniciais não se difundam para as outras fases de desenvolvimento, as atividades de teste devem ser iniciadas assim que os elementos necessários para o teste estejam disponíveis.

Bastos et al. (2007), completa dizendo que também é necessário planejar o ambiente onde os testes serão realizados, criando uma massa de dados para o teste, o modelo de dados, a configuração dos softwares (*browsers*, sistemas operacionais, entre outros), enfim, toda a estrutura onde o teste será realizado, juntamente com as configurações do *hardware*.

Para Molinari (2003) o teste deve possuir planejamento (gerando um plano de teste), projeto dos casos de teste (situações possíveis de teste), execução e avaliação dos testes. Paula Filho (2001) também acredita que planejar e desenhar cuidadosamente os testes é necessário, e que essas atividades devem ser cumpridas por pessoas que tenham experiência para poderem analisar os resultados dos testes.

Diversas técnicas de teste estão sendo aplicadas para encontrar erros, segundo Delamaro, Maldonado e Jino (2007). Porém para que haja benefícios para as empresas, é necessário utilizar uma ferramenta que faça o planejamento das atividades de testes.

3. Testes Automatizados

A automação de testes é um processo de testes híbrido, já que testes manuais por hábito são indispensáveis. Os testes manuais consistem em testes onde cada passo é descrito detalhadamente. Sua execução é feita de forma que o profissional de teste simule um usuário comum interagindo direto com a aplicação (informando os dados manualmente, clicando nos botões etc.).

A execução manual de um caso de teste se torna uma função rápida, porém a execução e repetição de um inúmero conjunto de testes é uma tarefa muito dispendiosa e cansativa.

Neste contexto é imprescindível a utilização de ferramentas de automação de execução de testes. Segundo Molinari (2010) os testes automatizados utilizam uma ferramenta que copia a interação com a aplicação tal qual um humano faria, porém com algumas limitações. A ferramenta chama os *scripts* de testes que já estão previamente criados e os executam.

Estes *scripts* que contêm os dados de entrada e as ações que deverão ser executadas sem interferência do profissional de testes, ou seja, testes significativos do caso de teste não serão ignorados por falha humana e facilitarão a identificação de um possível comportamento não desejado.

Por se tratar de um trabalho maçante, repetitivo e que exige tempo e organização por parte dos testadores, as organizações estão investindo na automatização dos testes, utilizando ferramentas apropriadas para tal função. Com a utilização destas ferramentas, pode-se identificar erros com mais eficiência.

Através da análise desta tabela, Bartié (2002) afirma que os testes automatizados ao logo do seu tempo de uso são mais econômicos, rápidos e eficientes do que os testes manuais. Acredita-se, segundo Moreira Filho e Rios (2003), que a automatização de testes significa a substituição do testador. No entanto, a automatização é apenas uma maneira de tornar os testes mais eficientes e confiáveis, representando um apoio à sua execução, pois é fundamental o papel do testador que cria e atualiza os *scripts*.

Com a automatização, os testes tornam-se mais precisos e encontram-se rapidamente as falhas, pois os *scripts* podem ser rodados a qualquer momento, repetindo os mesmos testes, algo que seria um trabalho cansativo para o testador.

Existem várias ferramentas de execução automática de testes, cada uma se que se adapta melhor ao tipo de teste a ser automatizado. O sucesso da automação de testes está diretamente ligado com a documentação dos casos de testes que serão automatizados e com as ferramentas que serão utilizadas no processo de automatização.

3.1. TestComplete

Segundo a AutomatedQA Corporation, o TestComplete é uma ferramenta utilizada para automação de testes para aplicações Windows e .NET, porém mesmo que a ferramenta seja uma aplicação Windows, tanto o sistema operacional quanto o tipo do servidor web testado não importam, pois a ferramenta suporta servidores de diversas plataformas (Linux, Windows, etc.).

A escolha desta ferramenta se deu, pois a mesma não depende de qualquer outra ferramenta de desenvolvimento. Obtém da gravação de uma série de comandos que simulam as ações do utilizador, a integração com os navegadores é de fácil execução e principalmente, pois esta ferramenta apóia plenamente *front-end* de testes web garantindo que a funcionalidade e a confiabilidade dos sites e aplicações que irão ao ar.

O TestComplete é aplicado a testes de unidade, funcional e de regressão. Pressman (2006) ressalta que o aumento das exigências dos usuários e da concorrência está aumentando cada vez mais na área de desenvolvimento de software, o que faz com que as empresas queiram aperfeiçoar seus processos.

No entanto, considerando a complexidade envolvida nos sistemas atuais, para se testar adequadamente um software, uma organização gasta 40% do esforço de projeto total em teste. Para diminuir o tempo e custo do processo de teste de software esta ferramenta propõem auxiliar os colaborados nas atividades relacionadas a este processo, utilizando as técnicas existentes de automação de teste que será abordado logo abaixo.

4. Desenvolvimento da Metodologia

A solução utilizada para o desenvolvimento dos testes automatizados foi a Nota Fiscal Eletrônica (NFS-e) do produto Flye-Nota. Utilizou-se para dar maior enfoque ao trabalho, a tela de emissão de nota fiscal.

A nota fiscal eletrônica, segundo a Receita Federal é um modelo nacional de documento fiscal eletrônico que substitui a atual emissão de documento fiscal em papel, simplificando as obrigações acessórias dos contribuintes e permitindo ao mesmo tempo o acompanhamento em tempo real das operações comerciais pelo Fisco.

Esta ferramenta apresenta muitas vantagens às empresas que utilizam este sistema, como a agilidade, redução de custos com impressão, acompanhamento em tempo real da tramitação das informações, ética e transparência na documentação de uma operação de circulação de mercadorias ou prestações de serviços.

Optou-se por esta escolha, pois a NFS-e sendo um sistema Web que possui muitos acessos e tarefas simultâneas, sendo de extrema importância que este não possua defeito algum, sendo assim sua automatização trará maior confiabilidade à empresa e aos contribuintes.

A fase de levantamento de requisitos da solução pode ser classificada de diversas formas, tendo a finalidade de compreender a relação entre objetos, tarefas e as próprias funções do sistema. Os requisitos funcionais descrevem as funções que o produto deverá realizar em benefício dos clientes.

Levando em consideração os pontos citados acima, foi então elaborado um ciclo de vida para os testes desta metodologia. Um ciclo de vida de teste é necessário para estruturar várias atividades, suas ordens e dependências. O mesmo foi criado contendo cinco fases distintas, são elas a Versão testável, Teste de integração, Teste de sistema, Teste de regressão e Teste de aceitação, cada um contendo suas características específicas.

A técnica utilizada neste trabalho para a composição da base de teste foi a de gerar dados conforme o conhecimento do especialista. Utilizando a técnica de derivação funcional, que une especificações de requisitos, análise e projeto, os casos de teste foram conduzidos na interface do software.

Esta técnica, conforme Rocha, Maldonado e Weber (2001), são empregados para demonstrar que as funções do software estão operando corretamente, que a entrada é adequadamente aceita e a saída é corretamente produzida e que a integridade da informação externa (uma base de dados, por exemplo) é conservada.

A organização da base de teste foi projetada envolvendo várias possibilidades das funções do software, onde se fez necessário elaborar pequenas situações para cada componente da tela, para que os mesmos fossem testados individualmente. Deste modo, criou-se situações válidas e inválidas.

4.1. Matriz de Cobertura

A matriz de rastreabilidade, também conhecida como *Traceability Rastreability Matrix (TRM)*, se trata de um documento de acompanhamento que facilita a visualizacao dos relacionamentos entre requisitos e outros artefatos ou objetos. A rastreabilidade entre requisitos e casos de teste permite, entre outras coisas, visualizar quantos casos de testes foram alocados para um requisitos e se algum dos mesmos não possui casos de testes alocados.

O cubo mágico, como também é conhecida a matriz de rastreabilidade, foi construindo utilizando o Microsoft Excel e o mesmo é constituído de uma planilha principal que possui os requisitos não funcionais dos testes, uma segunda com o planejamento funcional e outras oito planilhas que possuem os casos de testes que são relacionados a planilha principal.

A planilha principal é formada por requisitos não funcionais do sistema, sendo estes, restrições e comportamentos que o software deve satisfazer, e que compõem as três baterias de testes criadas. Para garantir a cobertura de testes deste trabalho foi utilizada a bateria 01, conforme figura 01.

Os requisitos escolhidos para formar cada bateria foram: a versão do Java utilizada na estação de trabalho, os browses utilizados para a execução dos testes, o sistema operacional instalado e se utilizada máquina virtual. Este primeiro documento do cubo mágico possui um link para acesso direto às planilhas de casos de teste. É apartir desta planilha que será possível verificar os casos de testes planejados e executados pela ferramenta de automação.

O planejamento funcional está contido na segunda planilha da matriz de rastreamento. Este documento abrange os testes de integração, testes de sistema e teste de regressão, que foram estruturados para serem cumpridos dentro de um cronograma de três semanas, elaborados com o conhecimendo especialista. Definido os requisitos para a

validação em conjunto com as baterias de testes, o mais importante dentre eles é a emissão de nota fiscal eletrônica, que será o alvo da composição deste trabalho. Pode ser verificado na figura 02.

Figura 01 - Bateria 01 da aba principal da matriz de rastreabilidade.

Bateria 01				
Ciclo	Versão do Java	Browsers	Sistema Operacional	Máquina Virtual
TRM 01	6.0	Internet Explorer	Windows	Nenhuma
TRM 02	6.0	Internet Explorer	Windows	Nenhuma
TRM 03	6.0	Internet Explorer	Windows	Nenhuma
TRM 04	6.0	Internet Explorer	Windows	Nenhuma
TRM 05	7.0	Internet Explorer	Windows	Nenhuma
TRM 06	7.0	Internet Explorer	Windows	Nenhuma
TRM 07	7.0	Internet Explorer	Windows	Nenhuma
TRM 08	7.0	Internet Explorer	Windows	Nenhuma

Fonte: Do autor

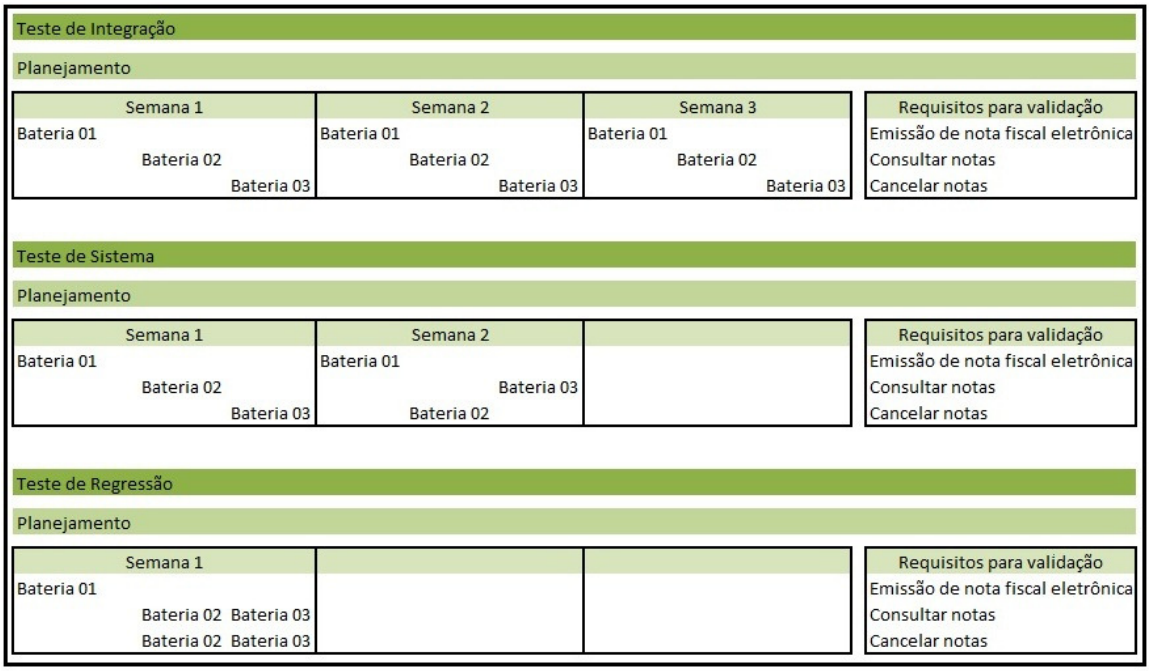
Para os testes de integração as baterias foram incorporadas às três semanas do planejamento, pois é nesta fase em que todos os componentes são unidos e que os testes devem cobrir todas as áreas integradas, dispendendo maior tempo para a execução dos testes.

Utilizando duas semanas do planejamento para a execução das baterias, o teste de sistema foi projetado para executar os testes em uma ordem que atenda a demanda da validação dos requisitos, verificando se estão de acordo com o que foi solicitado.

O teste de regressão será realizado em uma semana e as baterias de testes necessárias para suprir a qualidade foram incorporadas a ela. Este teste se torna menor pois o seu objetivo é averiguar se as funcionalidades já existentes foram afetadas com as mudanças realizadas no software.

As TRM criadas são diretamente ligadas a planillha principal da matriz de rastreabilidade. Nelas estão todos os casos de testes descritos de um conjunto de ações ou valores de entrada. Para cada bateria de teste formulada, os casos de testes contidos em uma TRM podem ser planejados e executados. Caso os testes tenham sido executados na ferramenta de automação *TestComplete* e realizados com sucesso a coluna “Passou” da planilha deve ser marcada juntamente com o usuário que testou na coluna “Quem testou”.

Figura 02 – Planejamento funcional da matriz de rastreabilidade.



Teste de Integração			
Planejamento			
Semana 1	Semana 2	Semana 3	Requisitos para validação
Bateria 01	Bateria 01	Bateria 01	Emissão de nota fiscal eletrônica
Bateria 02	Bateria 02	Bateria 02	Consultar notas
Bateria 03	Bateria 03	Bateria 03	Cancelar notas

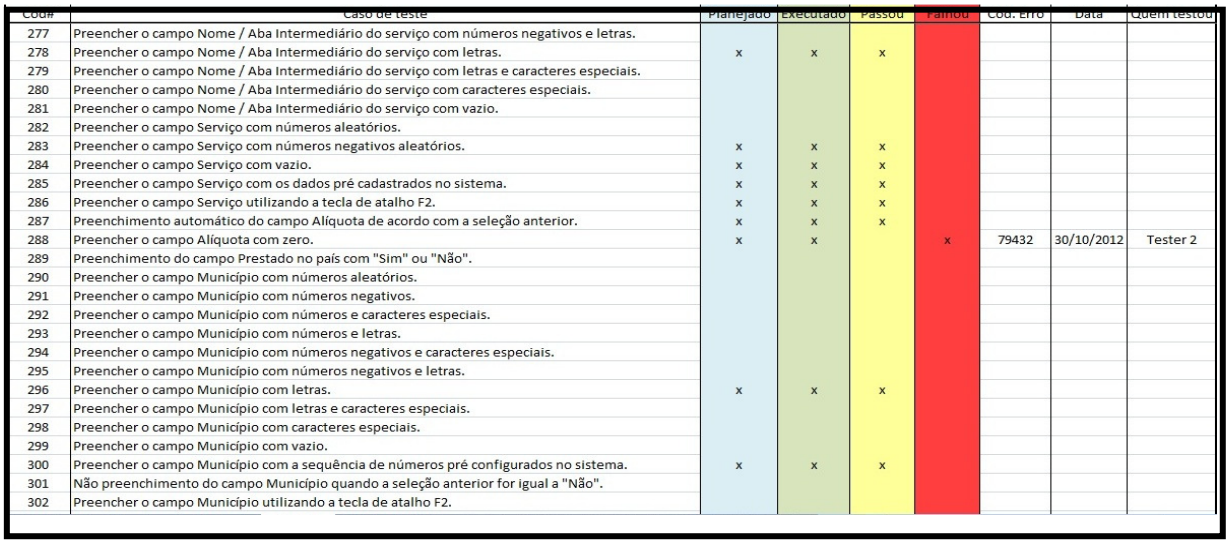
Teste de Sistema			
Planejamento			
Semana 1	Semana 2		Requisitos para validação
Bateria 01	Bateria 01		Emissão de nota fiscal eletrônica
Bateria 02	Bateria 03		Consultar notas
Bateria 03	Bateria 02		Cancelar notas

Teste de Regressão			
Planejamento			
Semana 1			Requisitos para validação
Bateria 01			Emissão de nota fiscal eletrônica
Bateria 02 Bateria 03			Consultar notas
Bateria 02 Bateria 03			Cancelar notas

Fonte: Do autor

Mas se o caso de teste não obtiver sucesso e encontrar uma falha a mesma deve ser marcada na coluna falhou, o erro deve ser salvo em uma ferramenta que reporte as falhas e o seu código deve ser marcado na coluna “Cód. erro”, juntamente com a “Data” e “Quem testou”. A TRM 01 é utilizada para especificar os casos de teste planejados e executados na emissão de nota fiscal eletrônica. Pode-se observar que um erro foi reportado pela ferramente de automação e sua erro foi devidamente cadastrada, conforme figura 03.

Figura 03 – TRM de casos de teste.



Cod#	Caso de teste	Planejado	Executado	Passou	Falhou	Cod. Erro	Data	Quem testou
277	Preencher o campo Nome / Aba Intermediário do serviço com números negativos e letras.							
278	Preencher o campo Nome / Aba Intermediário do serviço com letras.	x	x	x				
279	Preencher o campo Nome / Aba Intermediário do serviço com letras e caracteres especiais.							
280	Preencher o campo Nome / Aba Intermediário do serviço com caracteres especiais.							
281	Preencher o campo Nome / Aba Intermediário do serviço com vazio.							
282	Preencher o campo Serviço com números aleatórios.							
283	Preencher o campo Serviço com números negativos aleatórios.	x	x	x				
284	Preencher o campo Serviço com vazio.	x	x	x				
285	Preencher o campo Serviço com os dados pré cadastrados no sistema.	x	x	x				
286	Preencher o campo Serviço utilizando a tecla de atalho F2.	x	x	x				
287	Preenchimento automático do campo Alíquota de acordo com a seleção anterior.	x	x	x				
288	Preencher o campo Alíquota com zero.	x	x		x	79432	30/10/2012	Tester 2
289	Preenchimento do campo Prestado no país com "Sim" ou "Não".							
290	Preencher o campo Município com números aleatórios.							
291	Preencher o campo Município com números negativos.							
292	Preencher o campo Município com números e caracteres especiais.							
293	Preencher o campo Município com números e letras.							
294	Preencher o campo Município com números negativos e caracteres especiais.							
295	Preencher o campo Município com números negativos e letras.							
296	Preencher o campo Município com letras.	x	x	x				
297	Preencher o campo Município com letras e caracteres especiais.							
298	Preencher o campo Município com caracteres especiais.							
299	Preencher o campo Município com vazio.							
300	Preencher o campo Município com a sequência de números pré configurados no sistema.	x	x	x				
301	Não preenchimento do campo Município quando a seleção anterior for igual a "Não".							
302	Preencher o campo Município utilizando a tecla de atalho F2.							

Fonte: Do autor

4.2. Automatização dos Testes

Planejar a descrição precisa do comportamento esperando de um teste, por diversas vezes se torna algo muito difícil. Conforme Pezzé e Young (2008) quando muitas características de um programa com interface gráfica podem ser elencadas de uma forma adequada para auto-teste, algumas propriedades requerem uma pessoa para interagir com o programa e julgar seu comportamento. Não sendo possível evitar completamente o envolvimento humano na execução dos testes, pode-se pelo menos evitar a repetição desnecessária.

Para a automação dos testes da emissão de notas fiscais, foi utilizada a técnica de programação de *scripts* Captura e Reprodução *(Capture-Replay).* Este método grava (*Capture)* as ações do usuário interagindo com a aplicação, gerando um *script* de teste que pode ser reproduzido *(Replay).*

O processo de automação será executado conforme os casos de testes planejados na matriz de rastreabilidade e o primeiro passo para esta implementação foi a criação de um projeto no programa TestComplete utilizando a linguagem Java, pois a solução a ser testada é desenvolvida nesta linguagem.

Dentro deste projeto, deve-se acionar o botão de gravação de *scripts*, que em seguida já gravará todas as ações executadas. O browse do Internet Explorar é então aberto e o login para o acesso à página de emissão de notas é feito. Na sequencia um dos casos de testes planejados no cubo mágico é gravado e após esta ação o botão *Stop*da caixa de diálago *Recording*é executado.

Após a gravação de um caso de teste, a ferramenta cria um novo procedimento e converte todas as instruções realizadas na linguagem do *script* alvo, neste caso em Java. Todas as ações são transformadas em instruções de fácil entendimento, contribuindo assim para a criação e manutenção dos *scripts.*

Após um script ser gravado, o mesmo pode ser executado outras vezes onde a ferramenta executará todas as ações da mesma forma e ordem em que foram gravadas. Por este motivo há a necessidade de as gravações serem em ordem, para que as execuções não falhem.

Ao término das gravações o programa exibe um relatório de execução que apresenta algumas informações sobre a execução dos testes, erros ocorridos, ações realizadas entre outras, que é chamada de *Test Log*.

5. Conclusão

A necessidade de criação de softwares com grande qualidade aumentou a importância das atividades de teste de software. Com o avanço tecnológico as técnicas de teste de software puderam evoluir, dentre elas a automação dos testes.

Através do desenvolvimento da metodologia com a criação do ciclo de vida do software, juntamente com a matriz de rastreabilidade que fez os cruzamentos das baterias de testes dos requisitos não funcionais com os requisitos de validações e os casos de testes criados, juntamente com a automatização dos testes utilizando uma solução *WEB* foi possível confirmar o planejamento da proposta deste trabalho.

Através de ferramentas especializadas, a automatização surgiu com o intuito de beneficiar a produtividade dos projetos de desenvolvimentos. Existem várias ferramentas disponíveis. Dentre elas, as ferramentas de gestão e de execução de testes, foco de estudo deste trabalho.

Muitos dos casos de testes ainda requerem execução manual, porém os testes que podem ser automatizados apresentam considerável ganho de produtividade principalmente em relação ao tempo de execução. Com a realização deste trabalho, pode-se demonstrar que novas soluções podem contribuir e agilizar a fase de testes de software, considerando que é uma das fases mais importantes no ciclo de desenvolvimento quando se fala em qualidade.

Não se tem dúvida de que a automatização de testes é uma solução que alcança a agilidade necessária, porém também demanda custos, e tempo para a criação e manutenção dos scripts. Por este motivo, muitas vezes esta solução não é adotada pelas empresas de software. Através da ferramenta desenvolvida buscou-se diminuir o tempo de implementação destes scripts, gerando automaticamente os testes, para que posteriormentesejam executados por ferramentas adequadas, no caso a ferramenta *TestComplete*.

Referências

AUTOMATEDQA CORPORATION. GettingstartedwithTestComplete 9. [S.l.], [2012]. Disponível em:
<http://downloads.smartbear.com/Docs/Getting_Started_With_TestComplete.pdf>.
Acesso em: 02 outubro de 2012.

BARTIÉ, Alexandre. **Garantia da Qualidade de Software.** Rio de Janeiro: Elsevier, 2002.

BASTOS, Anderson et al. **Base de conhecimento em teste de software.** 2 ed. São Paulo: Martins, 2007.

BRAUDE, Eric. **Projeto de Software.** Porto Alegre: Bookman, 2005.

CAETANO, Cristiano. **Introdução à Automação de Testes Funcionais.** Disponível em: < http://www.testexpert.com.br/?q=node/178> . Acesso em: 26/06/2012.

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

MYERS, G. J. et al**.** The Artof Software Testing. [S.l.]:Wiley, 2004.

MOLINARI, Leonardo. **Testes de software**: produzindo sistemas melhores e mais confiáveis. 2 ed. São Paulo: Érica, 2003.

MOLINARI, Leonardo. **Inovação e Automação de testes de software**. São Paulo:
Editora Érica Ltda, 2010. 140 p.

MOREIRA FILHO, Trayahú R.; RIOS, Emerson. Projeto & engenharia de software: teste de software. Rio de Janeiro: Alta Books, 2003.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software**: fundamento, métodos e 140 padrões. 2 ed. Rio de Janeiro: LTC, 2001.

PRESSMAN, R. S. **Engenharia de Software.** 6. ed. Rio de Janeiro: McGraw-Hill,2006, 720p.

PEZZÉ, Mauro; YOUNG, Michael. **Teste e Análise de Software**: processos, princípios e técnicas. Porto Alegre, 2008.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software.** 2 ed. Rio de Janeiro: Alta Books, 2006.

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

SOFTEX, 2009 ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DOSOFTWARE BRASILEIRO – SOFTEX. MPS.BR – Guia Geral:2009. Disponível em: www.softex.br.