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

MARCELO DEHON BATISTA DE PRÁ

O USO DAS TÉCNICAS DATA-DRIVEN E KEYWORD-DRIVEN NO PROCESSO DE DESENVOLVIMENTO DOS SCRIPTS DE AUTOMAÇÃO DE TESTES FUNCIONAIS DE SOFTWARE

CRICIÚMA
2012

MARCELO DEHON BATISTA DE PRÁ

O USO DAS TÉCNICAS DATA-DRIVEN E KEYWORD-DRIVEN NO PROCESSO DE DESENVOLVIMENTO DOS SCRIPTS DE AUTOMAÇÃO DE TESTES FUNCIONAIS DE SOFTWARE

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

Orientadora: Prof.(ª) MSc Ana Claudia Garcia Barbosa.

CRICIÚMA

2012

MARCELO DEHON BATISTA DE PRÁ

O USO DAS TÉCNICAS DATA-DRIVEN E KEYWORD-DRIVEN NO PROCESSO DE DESENVOLVIMENTO DOS SCRIPTS DE AUTOMAÇÃO DE TESTES FUNCIONAIS DE SOFTWARE

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

Criciúma, 26 de novembro de 2012

BANCA EXAMINADORA

Profa. MSc. Ana Claudia Garcia Barbosa (UNESC) - Orientador

Prof. MSc. Gustavo Bisognin (UNESC)

Prof. Esp. Gilberto Vieira da Silva (UNESC)

Dedico este trabalho a todos que me apoiaram e incentivaram durante toda essa jornada, em especial a minha família.

AGRADECIMENTOS

Gostaria de agradecer primeiramente aos meus pais, que sempre me apoiaram desde o início para que um dia eu conseguisse chegar até esse momento.

Agradeço também a todos os professores do curso de Ciência da Computação, em especial a minha professora orientadora Ana, que me auxiliou durante todo esse projeto.

E também agradecer a todos que de certa foram contribuíram para que esse sonho se tornasse realidade.

“Você pode encarar um erro como uma besteira a ser esquecida, ou como um resultado que aponta uma nova direção”.

Steve Jobs

RESUMO

Esse projeto procura demonstrar algumas das principais técnicas de automação de testes, e desenvolver uma ferramenta que auxilie na criação e execução de casos de testes automatizados, baseando-se nas técnicas Data-Driven e Keyword-Driven para a criação dos scripts de execução dos testes. Inúmeros estudos ressaltam a importância de um processo de teste efetivo para minimizar o custo e proporcionar maior qualidade no desenvolvimento de software. Os processos de teste mais efetivos iniciam antes de qualquer programa ser escrito. Neste contexto, testes podem ser planejados desde a etapa de requisitos. A execução manual de um teste é rápida e efetiva, mas a execução e repetição de um vasto conjunto de testes manualmente é uma tarefa demorada e cansativa. A automação de testes aumenta a produtividade e atinge em um tempo menor aquilo que em geral é repetitivo no ambiente de testes.

Palavras-chave: Automação de testes. Teste de Software. Qualidade de Software. Data-Driven. Keyword-Driven.

ABSTRACT

This project seeks to demonstrate some key techniques for test automation, and develop a tool that assists in the creation and execution of automated test cases, based on the technical Data-Driven and Keyword-Driven for the creation of test scripts execution . Numerous studies emphasize the importance of an effective testing process to minimize the cost and provide higher quality in software development. The most effective testing processes before starting any program be written. In this context, tests can be designed from step requirements. The manual execution of a test is quick and effective, but the execution and repetition of a wide range of tests manually is time consuming and tiring. The test automation increases productivity and achieves in a shorter time than what is generally repetitive in the test environment.

Keywords: Automation testing. Software Testing. Software Quality. Data-Driven. Keyword-Driven.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GUI	Graphical User Interface (Interface Gráfica do Usuário)
ISO	International Ornanization for Standardization (Organização Internacional para Organização)
UML	Unified Modeling Language (linguagem de modelagem unificada)

SUMÁRIO

1 INTRODUÇÃO

Teste de software é definido como um processo, ou uma série de processos, projetados para garantir que o código desenvolvido faça exclusivamente o que é pretendido, devendo também ser previsível e consistente, evitando surpresas aos usuários (MYERS, 2004, tradução nossa).

Alguns consultores defendem que o teste de software serve para reduzir o custo dos problemas. Os gestores defendem que investir em qualidade de software é importante para que a aplicação seja conforme as necessidades e expectativas da organização. Ao se investir em testes, investisse em prevenção de defeitos. Dessa forma, certifica-se que o produto que será entregue está de acordo com as especificações e necessidades. Como consequência, se ganha em maior satisfação do usuário final, melhora a imagem da empresa, maior redução das incertezas que cercam o software, redução do custo de manutenção em produção do produto entregue, entre outros.

Automatizar um teste significa utilizar algum software que imita a interação com a aplicação no que se refere ao teste tal qual um ser humano faria (com algumas limitações). A principal razão do uso e da disseminação da automação dos testes de software é justamente a urgência cada vez maior de realizar mais testes em menos tempo. As aplicações ficaram mais complexas ao mesmo tempo em que se passou a exigir maior qualidade do produto entregue (MOLINARI, 2010).

A automação de testes funcionais visa automatizar um ou mais casos de testes funcionais. Testes funcionais visam testar as principais funcionalidades de um sistema, no qual se destacam os testes de interface (MOLINARI, 2010).

Muitas empresas ao tentar implantar um processo de automação de testes, seja ela funcional ou não (testes de desempenho, caixa branca e etc...) dentro de seu ambiente de produção, fracassam nas fases iniciais durante o processo de implantação, por alguns motivos como: expectativas irreais que ocorrem geralmente quando os gestores e testadores deixam-se seduzir pelas

promessas dos fabricantes de software de automação, onde quando começam a utilizar as ferramentas percebem que nem sempre é tão fácil automatizar como prometia os fabricantes, porém não é algo inatingível. Também por não tratarem a automação de testes como um novo projeto. Onde muitas vezes ocorre dos gestores acreditarem erroneamente que um processo de teste manual, e um processo automatizado são as mesmas coisas.

Dessa forma, esse projeto de pesquisa demonstra algumas das principais técnicas de automação de testes, e tem como objetivo principal desenvolver uma ferramenta que auxilie na criação e execução de casos de testes automatizados, baseando-se nas técnicas *Data-Driven e Keyword-Driven* para a criação dos scripts de execução dos testes.

1.1 OBJETIVO GERAL

Desenvolver uma ferramenta para criação e manutenção de scripts de automação de testes funcionais de software, com base nas técnicas *Data-Driven* e *Keyword-Driven*.

1.2 OBJETIVOS ESPECÍFICOS

Segue abaixo os objetivos específicos desse projeto de pesquisa:

a) estudar os principais software de automação de testes disponíveis no mercado;
b) diminuir o esforço gasto com manutenção de um projeto de automação de testes;
c) desenvolver um software para auxiliar no desenvolvimento de scripts e de automação de testes, com base nas técnicas *Data-Driven* e *Keyword-Driven*;
d) desenvolver um software para a criação de casos de testes automatizados.

1.3 JUSTIFICATIVA

Controlar a qualidade dos produtos de software é um grande desafio devido à alta complexidade dos produtos e inúmeras dificuldades relacionadas ao processo de desenvolvimento, que envolve questões humanas, técnicas, burocráticas, de negócio e políticas. Idealmente, os sistemas de software devem não só fazer corretamente o que o usuário (cliente) precisa, mas também fazê-lo de forma segura, eficiente, escalável, serem flexíveis, de fácil manutenção e evolução.

Inúmeros estudos ressaltam a importância de um processo de teste efetivo para minimizar o custo e proporcionar maior qualidade no desenvolvimento de software. Os processos de teste mais efetivos iniciam muito antes de qualquer programa ser escrito. Neste contexto, testes podem ser planejados desde a etapa de requisitos.

A execução manual de um teste é rápida e efetiva, mas a execução e repetição de um vasto conjunto de testes manualmente é uma tarefa muito demorada e cansativa. É normal que os testadores não verifiquem novamente todos os casos a cada mudança significativa do código. Deste cenário que surgem os erros de software, trazendo prejuízos para as equipes de desenvolvimento que perdem muito tempo para identificar e corrigir os erros, e também prejuízos para o cliente que, entre outros problemas, sofrem com constantes atrasos nos prazos combinados e com a entrega de software de qualidade duvidosa (DUSTIN, 2003).

Desenvolvimento de software é uma tarefa complexa que exige conhecimento técnico, organização, atenção, criatividade e também muita comunicação. É previsível que, em alguns momentos do desenvolvimento, em algumas das milhares linhas de código, alguns destes ou requisitos falhe, mas é imprevisível o momento no qual eles irão falhar.

Por isso, é imprescindível que exista uma maneira fácil e ágil de executar todos os testes em qualquer instante, e isso só é viável com o auxílio de testes automatizados (CHEQUE, 2011).

A automação dos testes aumenta a segurança da equipe para fazer alterações no código, seja por manutenção, refatoração ou até mesmo para adição de novas funcionalidades. Além disso, representar casos de teste através de programas possibilita a criação de testes mais elaborados e complexos, que poderão ser repetidos identicamente inúmeras vezes e a qualquer momento. Consequentemente, a quantidade de tempo gasto com a verificação do sistema aumenta, enquanto diminui o tempo gasto com a identificação e correção de erros, já que eles tendem a ser encontrados com antecedência. À medida que o tempo passa, o sistema vai crescendo e os programadores vão esquecendo certos detalhes de implementação, por isso quanto mais tarde um erro for encontrado, mais trabalhosa será a depuração para a sua localização.

Alguns pontos podem ser citados como casos de sucesso da implantação da automação de testes nas empresas, como: redução de custos, testes aprimorados, resultado precisos, diminuição do ciclo de vida do desenvolvimento e processo pronto para próximos projetos (SCALDAFERRI, 2010).

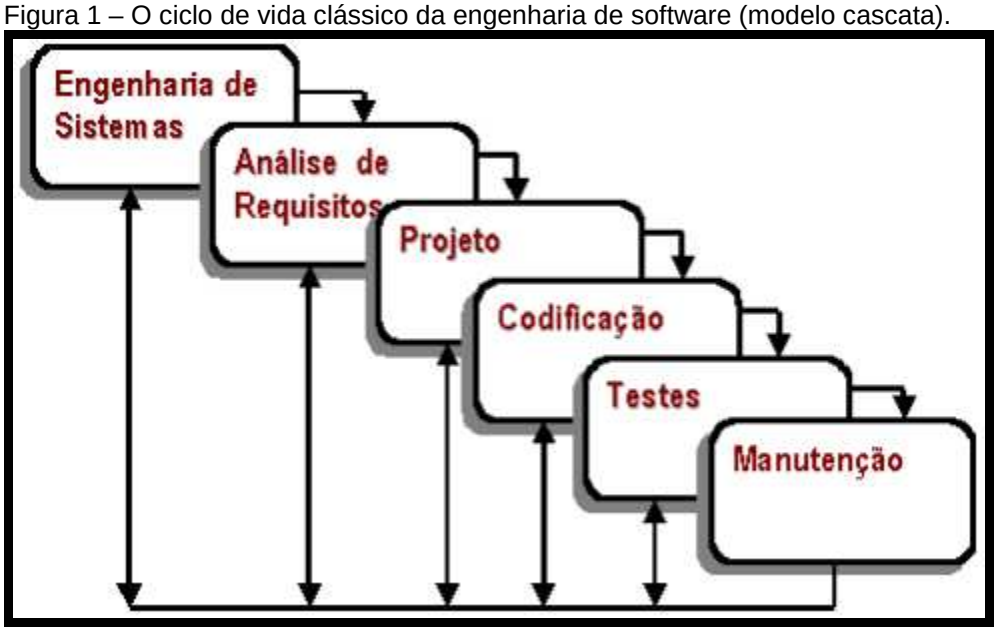
2 ENGENHARIA DE SOFTWARE

Engenharia de software é uma disciplina da engenharia voltada aos aspectos de produção de software, iniciando no estágio de especificação do sistema e indo até a manutenção desse sistema depois que ele entrou em operação (SOMMERVILLE, 2003).

A engenharia de software abrange um conjunto de três elementos principais, sendo eles: métodos, ferramentas e procedimentos. A união desses elementos é o que da possibilidade ao gerente de software o controle do processo de desenvolvimento oferecendo assim aos profissionais envolvidos uma base para a construção de um software de alta qualidade e produtividade (PRESSMAN, 1995).

A figura 1 ilustra um dos ciclos de vida mais comum utilizado na engenharia de software, também conhecido como modelo cascata. Este demonstra de uma forma sequencial o ciclo de desenvolvimento de um software, iniciando no nível de sistema e avançando ao longo da análise, projeto, codificação, teste e manutenção (PRESSMAN, 1995).

Figura 1 – O ciclo de vida clássico da engenharia de software (modelo cascata).



Fonte: Pressman (1995)

Além do modelo cascata, existem outros modelos de desenvolvimento, onde para cada ambiente deve ser utilizado o modelo que melhor se encaixa. Exemplos:

a) Modelo Queda d'Água;
b) Prototipação;
c) Desenvolvimento Interativo;
d) Modelo Espiral, entre outros.

2.1 REQUISITOS DE UM SOFTWARE

Os requisitos são as características que definem os critérios de aceitação de um produto. O processo de desenvolvimento de um software tem como principal objetivo colocar no produto as características que foram definidas como requisitos. Durante o desenvolvimento outras características podem aparecer, porém o software não deve ser projetado para incluí-las, pois normalmente todas as características extras significam um custo adicional ao projeto (PAULA FILHO, 2003).

Os requisitos definem basicamente o que software deve fazer, porém existem dois pontos de vistas diferentes a esse respeito: primeiramente existe o ponto de vista do cliente, que detalha a forma como ele realiza o seu trabalho, e como o software deve lhe auxiliar. E por outro lado existe o ponto de vista do desenvolvedor, que questiona a maneira como internamente o software deve funcionar. Com base nisso pode-se dividir os requisitos em (KOSCIANSKI; SOARES, 2006):

a) **requisitos funcionais:** descreve as funcionalidades que se espera do software quando este estiver pronto, como entradas, saídas, exceções etc;
b) **requisitos não funcionais:** descrevem restrições ao software de uma forma geral. Um exemplo de um requisito não funcional poderia ser exemplificado pelo tempo que um determinado

sistema de uma agência bancária leva para gerar o saldo da conta de um cliente.

2.2 QUALIDADE DE SOFTWARE

Existe uma série de atributos associados que refletem na qualidade de software, não sendo somente associados diretamente com o que o software faz mais sim também com o seu comportamento em geral, tais como: funcionamento, estrutura e organização do programa, tempo de resposta, documentação associada, entre outros (SOMMERVILLE, 2003).

Geralmente a qualidade de um produto é ligada diretamente com a qualidade do processo utilizado na produção. Muitas vezes por falta de ferramentas adequadas, por falta de qualificação das pessoas, por prazos curso e entres outros, levam os gerentes de projetos a eliminarem etapas relacionadas com a garantia da qualidade (PAULA FILHO, 2003).

Do ponto de vista do produtor a qualidade é o cumprimento dos requisitos, já do ponto de vista do consumidor, a qualidade pode ser definida como o atendimento às suas necessidades (BASTOS et al, 2007).

2.2.1 ISO/IEC 9126-1

A norma ISO/IEC 9126-1 descreve um modelo de qualidade do produto de software, envolvendo a qualidade do processo de desenvolvimento (características internas e externas), e também a qualidade em uso, que consiste na qualidade do software em condições reais de funcionamento.

A qualidade do processo de desenvolvimento pode ser dividida em seis características, sendo elas:

a) funcionalidade;

b) confiabilidade;

c) usabilidade;

d) eficiência;

e) manutenebilidade;

f) portabilidade.

Já a qualidade em uso pode ser divida em outras quatro características, sendo elas:

a) eficácia;

b) produtividade;

c) segurança;

d) satisfação.

A norma ISO/IEC 9126-1 não apresenta métricas para medir a qualidade das características citadas acima, e sim propõe que cada empresa desenvolva suas próprias métricas para saber se cada característica está dentro ou não de seus padrões aceitáveis de qualidade.

3 TESTE DE SOFTWARE

A atividade de teste de software é um elemento crítico da garantia de qualidade de software e representa a última revisão de especificações, projeto e codificação de um software (PRESSMAN, 1995).

O teste consiste em executar um programa com a intenção de encontrar erros (*bugs*) (MYERS, 1979, tradução nossa).

Embora as revisões técnicas possam ser mais eficazes para detectar e remover defeitos, os testes têm a importância de aumentar as revisões, e consequentemente aumentar a qualidade (PAULA FILHO, 2003).

Assim, a tarefa de teste de software é qualquer atividade destinada a avaliar um atributo ou capacidade de um programa ou sistema, verificando se este cumpre cos seus resultados esperados.

3.1 OBJETIVOS DA ATIVIDADE DE TESTE

O objetivo principal do processo de teste é o de encontrar a maior quantidade possível de defeitos existentes no software, servindo também para ajudar a definir o escopo do projeto de teste (BASTOS, 2007).

Myers estabelece uma série de regras que podem servir como objetivos da etapa de teste de um software, tais como:

a) a atividade de teste é o processo de executar um programa como o objetivo de descobrir um erro;
b) um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto;
c) um teste bem sucedido é aquele que revela um erro ainda não descoberto.

Se uma atividade de teste for conduzida com sucesso, ela tenderá a descobrir erros no software. Como um resultado secundário ao descobrimento de erros, ela também irá demonstrar que as funções para as quais o software foi desenvolvido estão de acordo com suas especificações funcionais, e estão funcionando corretamente. Porém a atividade de teste não pode mostrar a

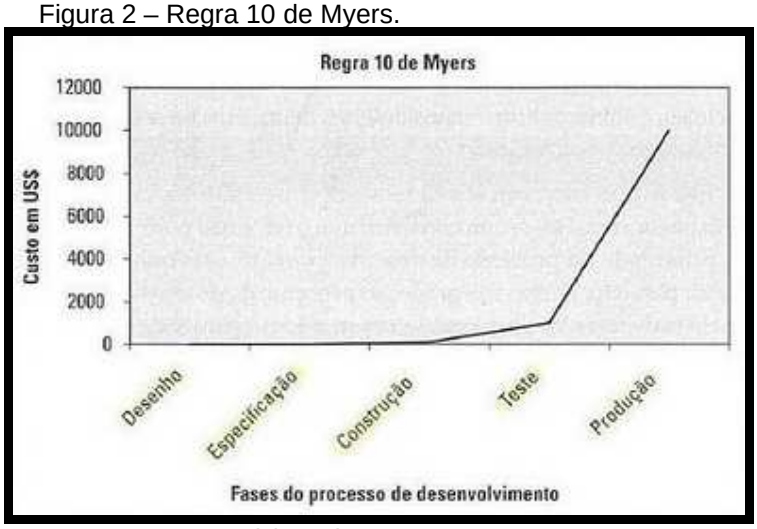
ausência de erros, ela só pode mostrar que os defeitos estão presentes (PRESSMAN, 1995).

3.2 O CUSTO DOS DEFEITOS (REGRA 10 DE MYERS)

Os defeitos existentes em um software, na maioria das vezes, constituem-se em riscos tanto para os negócios, como também para a imagem da empresa (BASTOS et al, 2007).

A figura 2 demonstra a Regra 10 de Myers, onde é observado que o custo de um defeito tende a ser maior, quando mais tarde esse defeito for encontrado. Defeitos encontrados durante o processo de produção do software tendem a custar muito mais que defeitos encontrados nas fases de especificação.

Figura 2 – Regra 10 de Myers.



Fonte: Bastos et al (2007).

No livro *The Art of Software Testing* lançado em 1979 por Myers, o mesmo afirma que (MYERS, 1979, tradução nossa):

a) testes unitários podem remover entre 30% e 50% dos defeitos;
b) os defeitos remanescentes dos testes unitários, podem ser removidos entre 30% e 50% com os testes de sistemas;
c) com isso, os sistemas podem ir para a produção com restando ainda aproximadamente 49% de defeitos;

d) e por último, as revisões de código pode reduzir entre 20% e 40% desses defeitos.

3.3 DIFERENÇA ENTRE DEFEITO E FALHA

Pode-se definir como um programa defeituoso, aquele que não funciona como deve. Geralmente um defeito é causado por um erro que faz o programa parar de funcionar, porém também existem defeitos menos aparentes onde o sistema continua funcionando. Já uma falha é o resultado errado provocado por um defeito ou alguma condição inesperada (KOSCIANSKI; SOARES, 2006).

Falhas ou defeitos podem existir, mas nem sempre serem visíveis. É possível que um programa funcione durante anos sem que ocorra uma falha, tudo dependerá das linhas de códigos executadas, e das condições esperadas pelas mesmas. (KOSCIANSKI; SOARES, 2006).

Figura 3 - Defeito x erro x falha.

Fonte: Dias Neto (2007).

A figura 3 demonstra quando e como geralmente ocorre um defeito, um erro ou uma falha.

4 PROCESSOS DE TESTE

O objetivo de um processo de teste, com metodologia própria, é minimizar os riscos causados por defeitos provenientes do processo de desenvolvimento (BASTOS et al, 2007).

Para cada etapa do processo de desenvolvimento, existe uma etapa de teste a ser realizadas. Com isso, o processo de teste pode-se dividir em alguns níveis, tais como:

a) teste unitário;

b) teste de integração;

c) teste de sistema;

d) teste de aceitação.

A figura 4 demonstra os níveis de teste numa relação entre escopo do software sob o teste e a realidade ambiente existente. Para cada nível de teste existem particularidades, tais como: configuração de hardware, configuração de software, interface, testadores etc. Desta forma, na medida em que o teste chega a níveis mais altos, o ambiente se torna mais realístico (CRAIG; JASKIEL, 2002, tradução nossa).

Figura 4 – Níveis de Testes versus Realidade de Ambiente.

Fonte: Oliveira (2007).

4.1 CONCEITO “V” DE TESTE

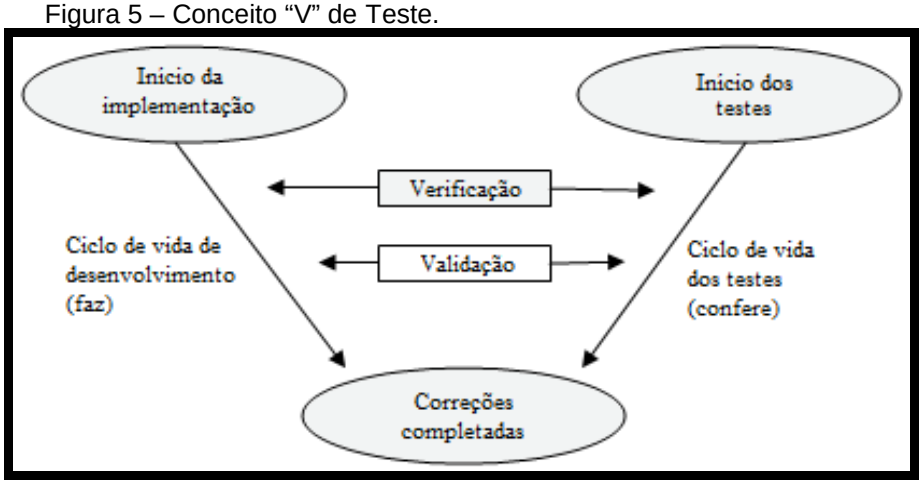
O ciclo de vida de testes pressupõe que sejam realizados testes ao longo de todo o processo de desenvolvimento. Pode-se dizer que o ciclo de

testes e de desenvolvimento são totalmente interdependente, porém o ciclo de teste é dependente da conclusão dos produtos do ciclo de desenvolvimento (BASTOS et al, 2007).

O modelo “V” de teste enfatiza as atividades de validação e verificação, com o intuito de prevenir/detectar defeitos, e minimizar os riscos do projeto (CAETANO, 2008). Os procedimentos de fazer e conferir devem trabalhar juntos até o fim do projeto, conforme pode ser visto na figura 5. Com isso, podem-se existir dois grupos distintos trabalhando simultaneamente, onde um trabalha com o objetivo de implementar o sistema (fazer), e outro com o objetivo de executar procedimentos de teste (conferir) (BASTOS et al, 2007).

A figura 5 mostra que os processos de desenvolvimento e de teste iniciam sempre no mesmo ponto, partindo sempre das mesmas informações iniciais.

Figura 5 – Conceito “V” de Teste.



Fonte: Bastos et al (2007).

4.2 TESTE UNITÁRIO

Tem como objetivo, a validação de um objeto logicamente tratado como uma unidade de implementação. Em linguagens orientadas a objetos, uma unidade é tipicamente uma classe (PAULA FILHO, 2003).

As atividades de testes de unidades normalmente são consideradas uma associação da etapa de codificação do software. Depois que o código

fonte é desenvolvido, revisado e verificado, inicia-se o projeto de casos de teste de unidade (PRESSMAN, 1995).

4.3 TESTE DE INTEGRAÇÃO

Após garantir que todos os componentes individuais estão funcionando corretamente, é necessário realizar os testes de integração entre esses componentes, a fim de saber o que causou uma falhar quando a mesma ocorrer (PFLEEGER, 2004). Segundo Pressman (1995), o objetivo do teste de integração, é garantir que a partir dos módulos testados individualmente no nível de unidade (teste unitário), consiga-se construir a estrutura do programa que foi determinada pelo projeto inicial.

As duas principais técnicas para a realização dos testes de integração são chamadas de *bottom-up* e *top-down*.

Através da abordagem *bottom-up*, o programa é desenvolvido a partir de rotinas de níveis mais baixos, onde estas dão suporte e prestam serviços as rotinas de níveis mais altos. Por exemplo, uma função para realizar a validação de CPF, pode ser chamada de vários pontos de um programa, com isso será então um dos primeiros pontos a serem desenvolvidos (KOSCIANSKI; SOARES, 2006).

Na abordagem *top-down*, faz-se o inverso, onde o programador trabalha supondo que o código de baixo nível já esteja pronto. Por exemplo, podem-se criar chamadas à verificação de CPF, mesmo sabendo que ela ainda não existe. Em seu lugar, pode haver uma rotina fantasma (*stub*), que apenas retorna sempre um valor verdadeiro (KOSCIANSKI; SOARES, 2006).

4.4 TESTE DE SISTEMA

O teste de sistema tem como objetivo principal por à prova o sistema baseado em computador (PRESSMAN, 1995). Diferentemente dos testes de unidade e integração que tinha como objetivo testar respectivamente

componentes individualmente, e em seguida juntos, o teste de sistema tem a finalidade de testar se todos os componentes quando integrados, vão realizar as funções aos quais à eles foram atribuídas (PFLEEGER, 2004; PRESSMAN, 1995).

4.5 TESTE DE ACEITAÇÃO

O teste de aceitação tem como objetivo comparar os requisitos inicias com a necessidade atual dos usuários finais, sabendo assim se o produto desenvolvido irá satisfazer as necessidades do mesmo. Geralmente é realizado por um grupo restrito de pessoas (usuários finais), em um ambiente muito parecido com aquele em que será o ambiente real de produção (MYERS, 2004, tradução nossa).

Esse tipo de teste deve ser projetado para determinar se o software foi desenvolvido ou parte de dele que está liberada para testes, estão aptos para serem utilizados pelo usuário final em ambiente de produção. Enquanto as outras etapas do processo de teste buscam encontrar defeitos no software, o teste de aceitação busca encontrar inconformidades com as regras de negócio, podendo assim ser classificado também como um tipo de teste funciona (BASTOS et al, 2007).

5 TÉCNICAS DE TESTE

Os procedimentos de testes devem ser amplos o suficiente para exercitar as funções do sistema. Como o teste busca encontrar a maior quantidade de defeitos possíveis, é importante saber onde os defeitos podem ser encontrados, uma boa prática é saber como os defeitos podem ser criados dentro do ambiente do software (PFLEEGER, 2004).

Levando em questão a complexidade de tamanhos dos softwares desenvolvidos atualmente, fica claro que é muito difícil de atingir 100% de cobertura de todos os requisito e linhas de códigos existentes. Com isso, as técnicas de testes têm como principal objetivo, auxiliar os analistas a identificar os casos de testes mais importantes, garantindo a maior cobertura possível dentro de um prazo determinado (CAETANO, 2008).

5.1 TESTE DE CAIXA BRANCA

A técnica de teste de caixa branca se baseia na estrutura de controle do projeto procedimental (código-fonte) para derivar casos de testes. Através dessa técnica o engenheiro de software consegue derivar casos de testes que garantem os seguintes testes (PRESSMAN, 1995):

a) testar todos os caminhos de códigos, garantindo que cada caminho possível tenha sido exercitado pelo menos uma vez;
b) exercitar todas as decisões lógicas para valores booleanos (falso ou verdadeiro);
c) executar todos os laços em suas fronteiras, dentro de seus limites operacionais;
d) executar estruturas de dados internas garantindo suas validades.

5.2 TESTE DE CAIXA PRETA

Os métodos de teste de caixa preta têm por principal objetivo ter os requisitos funcionais do software (PRESSMAN, 1995). São usados critérios

para a geração de casos de testes, com o objetivo de avaliar a aderência, ou conformidade do software desenvolvido com relação aos requisitos especificados (CAETANO, 2008). Dessa forma os testes funcionais tendem a garantir que os requisitos estão corretamente codificados (BASTOS et al, 2007).

Os testes de caixa preta são feitos através de fornecimento de dados de entradas pelo testador, onde este analisa as saídas correspondentes. Se as saídas não foram iguais às previstas, então significa que o teste encontrou um defeito no software. O principal problema para os responsáveis pelos testes é selecionar as principais entradas com maiores probabilidades de encontrar erros. Muitas vezes essas escolhas são feitas com base na experiência dos engenheiros de software, contudo existem algumas técnicas que podem auxiliar nessas escolhas, tais como (SOMMERVILLE, 2003):

a) particionamento de Equivalência;
b) análise de Valor Limite;
c) teste por tabela de decisão;
d) teste de matriz ortogonal;
e) teste de transição de estado.

Devido ao foco desse projeto de pesquisa, a seguir serão explicadas somente as técnicas *Particionamento de Equivalência* e *Análise de Valor Limite*, onde estar serão utilizadas no decorrer do projeto para o desenvolvimento do software final.

5.2.1 Teste Funcional

Teste funcional é uma técnica a partir da qual o software é avaliado conforme o ponto de vista do usuário. Por se tratar de um teste de caixa preta, são fornecidas as entradas e avaliadas suas respectivas saídas geradas, verificando assim se estão em conformidade com os requisitos funcionais do projeto (DELAMARO; MALDONADO; JINO, 2007).

Os testes funcionais são projetados visando a garantia de que os requisitos funcionais e as especificações do software tenham sido atendidas (BASTOS et al, 2007).

Testes funcionais bem projetados devem ter alta probabilidade de detectar defeitos. Com isso, definem-se algumas diretrizes que podem ser seguidas para a criação dos casos de testes, tais como (PFLEEGER, 2004):

a) ter grande probabilidade de detectar defeitos;
b) utilizar equipe de testes independentes dos projetistas e programadores;
c) ter um critério de encerramento;
d) testar entradas válidas e também as inválidas;
e) o teste funcional em hipótese alguma deve modificar o sistema para facilitar os testes;
f) conhecer as ações e as saídas esperadas.

Molinari (2008) define o teste funcional como o mais importante tipo de teste, pois é este que mostra se a aplicação funciona ou não em tudo aquilo para qual ela foi desenvolvida para atender, no que se refere aos requisitos funcionais.

5.2.2 Teste de Regressão

Para que uma aplicação cresça com integridade, é necessário sempre que for inserida ou alterada uma característica da aplicação, testar toda ela novamente ou uma parte significativa. Isso se faz necessário, pois ao se inserir algo novo, ou alterar algo já existente, pode ser que algo tenha sido estragado erroneamente (MOLINARI, 2008). O teste de regressão é importante por que as mudanças ou correções de erros, tendem a serem mais propensas a gerarem novos erros do que quando o código fonte foi criado pela primeira vez (MYERS, 2004, tradução nossa).

5.2.3 Particionamento de Equivalência

É uma técnica que tem como objetivo primário, reduzir a quantidade de casos de testes. Essa técnica se baseia em descobrir através de termos lógicos, conjuntos de situações que produzam ou gerem o mesmo comportamento na aplicação. Definindo as classes de equivalência, para efeito de teste, basta ter um caso de teste que represente a situação representada, para que o teste de todos os valores da classe de equivalência seja válido (MOLINARI, 2008).

Considerando que as regras de negócio de uma determinada empresa para definição da idade de contratação de um funcionário, sejam as seguintes:

a) **faixa etária 0-16:** não contrata o funcionário;

b) **faixa etária 16-18:** contrata o funcionário somente em tempo parcial;

c) **faixa etária 18-55:** contrata o funcionário em tempo integral;

d) **faixa etária 55-99:** não contrata o funcionário.

Considerando os valores acima, o código de aplicação (não há como saber) poderia ser implementado conforme demonstrado na figura 6.

Figura 6 – Particionamento de Equivalência (exemplo de código fonte).

```
Se (appIdade >= 0 && appIdade <= 16)
        contratacaoStatus = "NAO";
Se (appIdade >= 16 && appIdade <= 18)
        contratacaoStatus = "PARCIAL";
Se (appIdade >= 18 && appIdade <= 55)
        contratacaoStatus = "INTEGRAL";
Se (appIdade >= 55 && appIdade <= 99)
        contratacaoStatus = "NAO";
```

Fonte: Adaptado de Molinari (2008).

Como para efeitos de testes é na maioria das vezes fica inviável realizar todas as variações possíveis (0,1,2,3, ... , 99) de valores para idade dentro dos limites especificados, é definida algumas classes ou grupos onde cada um possui valores equivalentes. Onde testando as condições com apenas um desses valores estará testando indiretamente todos outros valores da classe de equivalência.

Considerando-se como exemplo o código fonte da figura 6, poderia se criar uma classe de equivalência com os valores “0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16”, onde qualquer um desses valores seriam válidos para realizar o teste da primeira condição (se idade maior ou igual a 0 (zero), e idade menor ou igual a 16) , não havendo a necessidade de testar todas as variações possíveis entre 0 e 16.

5.2.4 Técnica de Análise de Valor Limite

É uma técnica derivada da técnica de particionamento de equivalência, com isso esta também tem como objetivo reduzir a quantidade de casos de testes. A principal diferença dessa técnica para e particionamento de equivalência, é que essa parte do princípio que muitos problemas podem estar ao redor dos limites das classes de equivalência descobertas (MOLINARI, 2008).

Considerando as mesmas regras de negócio utilizada no exemplo anterior da técnica de particionamento de equivalência:

a) faixa etária 0-16: não contrata o funcionário;

b) faixa etária 16-18: contrata o funcionário somente em tempo parcial;

c) faixa etária 18-55: contrata o funcionário em tempo integral;

d) faixa etária 55-99: não contrata o funcionário.

Considerando o código fonte da figura 6, pode-se perceber que o mesmo possui um problema de fronteira quando a idade for igual a 16, onde a aplicação poderá assumir dois valores finais para contratação do funcionário.

Com isso, utilizando essa técnica, poderia se definir os seguintes valores para serem usados nos testes: [-1, 0, 1]; [15, 16, 17]; [17, 18, 19]; [54, 55, 56] e [98, 99, 100].

6 AUTOMAÇÃO DE TESTES FUNCIONAIS

Pode-se diferenciar a tarefa de teste de automação de teste da seguinte maneira. Na primeira, é realizada a tarefa de testar, e na segunda é a utilização de um software imitando um ser humano no que diz respeito à integração com a aplicação (FEWSTER; GRAHAM, 1999, tradução nossa).

A automação de testes aumenta em muito a produtividade e atinge em um tempo muito menor aquilo que em geral é repetitivo no ambiente de teste. Contudo os testes manuais não podem ser eliminados em hipótese alguma, sendo que os mesmos devem ser focados naquilo que é muito caro de se automatizar (MOLINARI, 2010).

Sendo assim, pode-se afirmar que o grande potencial da automação de testes funcionais, está nos testes de regressão. Pois basta automatizar uma vez um determinado teste, para que o mesmo possa ser repetido quantas vezes forem necessárias (MOLINARI, 2010).

Segundo Lages (2011), a automação de teste possui uma alta capacidade de reduzir esforços, porém em contrapartida quando não bem planejada, a mesma tende a possuir uma alta propensão a falhas, conforme pode ser observado na tabela 1.

Tabela 1 – Resultados de sucesso e insucesso da automação de testes.

SUCESSO	INSUCESSO
Redução de custos	Tempo desperdiçado
Testes melhorados	Dinheiro desperdiçado
Resultados precisos	Resultados imprecisos
Diminuição do ciclo de vida do desenvolvimento	Equipe desmoralizada
Processo pronto para novos projetos	Produtividade reduzida
	Oportunidade perdida

Fonte: Lages (2011).

6.1 HISTÓRIA DA AUTOMAÇÃO DE TESTES

Segundo Hayes (2009, tradução nossa), a evolução das ferramentas de automação de testes nos últimos 25 anos pode ser dividida da seguinte maneira:

a) **ferramentas de captura e execução de texto:** apenas capturavam os textos a ser executados dos mainframes e repetiam. Surgia assim o conceito de script de teste, porém esses não podiam ser alterados;

b) **ferramentas de captura e execução de texto com interface caractere:** com o surgimento dos PC torna-se possível manusear os scripts criados pelas ferramentas;

c) **ferramentas de captura da interface GUI e execução:** com o crescimento do PC, tornou-se possível também o crescimento das ferramentas de automação de teste. Com isso a interface GUI e todos os seus elementos passaram a ser detectados, podendo assim ser automatizados. Como consequência o uso de scripts para automação se tornou algo mais popular;

d) **frameworks customizáveis:** devido à mudança da plataforma de 16 bits para 32 bits, surgiu a necessidade da integração entre as ferramentas de automação. Devido a grande demanda pelo uso das ferramentas de automação, começou a surgir engenheiros de software especialistas nas ferramentas de automação de testes;

e) **frameworks comerciais:** os fabricantes passaram a integrar suas ferramentas com outras ferramentas do próprio fabricante, bem como também comprar ferramentas de outros fabricantes. Os frameworks passaram a ser vendidos apoiados por algum processo, fazendo com que os clientes ficassem presos aos fabricantes, muitas vezes limitando as empresas na evolução da qualidade de seus aplicativos.

6.2 TIPOS DAS FERRAMENTAS

Segundo Molinari (2010) as ferramentas de automação de testes podem ser classificadas em:

a) **ferramenta de planejamento de teste:** é um tipo de ferramenta fundamental em qualquer ambiente de teste, onde essa ajuda a projetar e planejar os testes, permitindo criar requisitos e casos de testes;

b) **GUI Teste Driver com Command Script:** é uma ferramenta utilizada para ajudar na execução dos testes que fazem uso de uma GUI (Graphical User Interface), onde as ações de testes são gravadas e após reexecutadas através de scripts. Possui conceitos de programação como loop e if/then/else;

c) **Gui Test Driver com Visual Script:** semelhante com o item anterior, diferenciando pelo fato do script não possuir uma linguagem, sendo completamente visual. Algumas até possui uma linguagem para dar suporte à inserção de alguma lógica no meio do script;

d) **Carga e Performance:** ferramentas especializadas em realizar testes com grandes quantidade de dados no sistema, podendo também simular virtualmente vários usuários simultâneos utilizando o mesmo sistema;

e) **Test Execution Managers (Gerenciadores de Execução de Testes):** ferramentas para gestão da automação de execução dos testes com uma interface gráfica;

f) **Code Coverage (Cobertura de Código):** ajuda a analisar a qualidade dos testes, analisando o código fonte em tempo de execução;

g) **Defect Tracking (Rastreamento de Defeito):** ferramentas que auxiliam no gerenciamento de um banco de dados dos defeitos e problemas encontrados nos testes;

h) **Unit Testing (Teste Unitário):** ferramentas que dão suporte aos testes unitários. Possuem uma interface que permite testar o software em nível mais próximo do código.

6.3 TÉCNICAS PARA CRIAÇÃO E MANUTENÇÃO DE SCRIPTS DE AUTOMAÇÃO DE TESTES

Segundo Graham e Fewster (1999, tradução nossa), existem cinco grandes técnicas para projetar e desenvolver scripts para as ferramentas de automação de testes:

a) scripts linerares;
b) scripts estruturados;
c) scripts compartilhados;
d) data-driven scripts;
e) keyword-driven scripts.

As técnicas acimas não exclusivas, ou seja, não devem ser utilizadas individualmente e sim em conjunto, onde cada um tem suas vantagens e desvantagens.

6.3.1 Scripts Lineares

Os scripts lineares são aqueles que implementam suas instruções de forma sequencial. Em muitos casos de testes estes scripts são criados através de ferramentas do tipo Record & Playback, as quais somente implementam a criação de scripts lineares (FEWSTER; GRAHAM, 1999, tradução nossa).

Geralmente esse tipo de script é simples e fácil de ser desenvolvido, porém por outro lado são muitos vulneráveis à mudança do software, tendo assim uma manutenção mais cara (MOLINARI, 2010)

6.3.2 Scripts Estruturados

Scripts estruturados são scripts que implementam estruturas de controle especiais (seleção ou interação) ou estrutura de chamada a outros scripts, portanto, estes scripts são programados, mesmo que parcialmente (FEWSTER; GRAHAM, 1999, tradução nossa).

A estrutura seleção avalia condições que determinam se um conjunto específico de instruções será executado. A estrutura interação possibilita repetir um conjunto de instruções uma determinada quantidade de vezes. Esta estrutura é utilizada, por exemplo, para a leitura de uma sequência de registros num arquivo de dados. Neste caso, a sequência pode ser executada até que seja concluída a leitura de todos os registros (FEWSTER; GRAHAM, 1999, tradução nossa).

Além disso, um script pode chamar outro *script*. Este técnica pode ser usada para dividir scripts grandes em scripts menores e mais gerenciáveis, melhorando o seu reuso (FEWSTER; GRAHAM, 1999, tradução nossa).

6.3.3 Scripts Compartilhados

Scripts compartilhados são scripts utilizados por mais de um caso de teste. O uso desta técnica visa identificar tarefas repetitivas que possam ser reutilizadas por mais de um caso de teste. Neste caso, são criados scripts que realizam tais tarefas, para que sejam chamados num ponto específico de um caso de teste (FEWSTER; GRAHAM, 1999, tradução nossa).

O compartilhamento destes scripts pode ser feito entre casos de teste de um mesmo sistema ou de diferentes sistemas. Entre as vantagens desta técnica, destaca-se principalmente a melhoria do reuso de *scripts*. Entre as desvantagens, destaca-se a existência de um número maior de scripts para serem mantidos (FEWSTER; GRAHAM, 1999, tradução nossa).

6.3.4 Data-Driven Scripts

A técnica data-driven (dirigido a dados) consiste em extrair dos scripts de teste os dados de teste, que são específicos por caso de teste, e

armazená-los em arquivos separados dos scripts de teste. Os scripts de teste passam a conter apenas os procedimentos de teste (lógica de execução) e as ações de teste sobre a aplicação, que normalmente são genéricos para um conjunto de casos de teste. Assim, os scripts de teste não mantêm os dados de teste no próprio código, obtendo-os diretamente de um arquivo separado, somente quando necessário e de acordo com o procedimento de teste implementado (FEWSTER; GRAHAM, 1999, tradução nossa).

6.3.5 Keyword-Driven Scripts

A técnica *Keyword-driven* (dirigido a palavras chaves) propõe separar o desenvolvimento dos casos de teste do desenvolvimento dos *scripts* de teste. Isto possibilita que testadores experientes no domínio da aplicação concentrem-se na criação dos arquivos de teste, enquanto que profissionais com conhecimento técnico concentrem-se nos scripts de suporte (FEWSTER; GRAHAM, 1999, tradução nossa).

O uso da técnica *Keyword-driven* diminui a complexidade dos *scripts*, pois, assim como ocorre quando descrevemos um teste manual, permite que os casos de teste sejam especificados de maneira menos detalhada (FEWSTER; GRAHAM, 1999, tradução nossa).

7 TRABALHOS CORRELATOS

O processo de teste cada vez mais vem sendo utilizado no ambiente de desenvolvimento de software das empresas como sendo uma das principais estratégias na busca e no controle da qualidade dos softwares desenvolvidos.

Junto ao teste de software, também se criou o conceito de automação de testes, que tem como principal objetivo reduzir esforços por parte do testador, quando se diz respeito a testes repetitivos.

A seguir são demonstrados alguns exemplos da utilização do teste de software, como também da automação de teste.

7.1 DESENVOLVIMENTO DE UMA METODOLOGIA APLICADA AO GERENCIMENTO E ACOMPANHAMENTO DE TESTE DE SOFTWARE VIA WEB

Este projeto foi desenvolvido por Fernando Bettiol Lopes no ano de 2008, como trabalho de conclusão de curso na Universidade do Extremo Sul Catarinense, propondo-se a criação de uma ferramenta para o gerenciamento e controle efetivo do teste de software com foco nos principais conceitos definidos pela técnica de caixa preta.

Com o aumento da complexidade dos sistemas atuais, começou-se a exigir uma avaliação cada vez mais criteriosa das funcionalidades dos produtos de software. Em virtude disso, começou-se a serem utilizadas ferramentas que detectam erros de forma automática como uma forma de auxiliar os engenheiros de software no processo de teste.

Dessa forma o autor do projeto desenvolveu uma ferramenta com o objetivo de realizar a ligação entre o sistema a ser testado e a ferramenta de teste automático, permitindo assim a integração entre os dois sistemas.

A metodologia aplicada na ferramenta desenvolvida permite o testador cadastrar todas as informações necessárias para o desenvolvimento e execução do seu plano de teste. A ferramenta permite também que sejam

cadastrados os usuários que fazem parte do grupo de teste, informando o papel de cada um dentro desse grupo.

Segundo o autor, a ferramenta desenvolvida conseguiu ser segura para a utilização, devido a sua função de verificação dos limites que cada usuário possui, e também permitiu que fossem cadastradas diversas características necessárias para a criação de um plano de execução de teste, podendo assim acompanhar a trajetória de um determinado erro.

7.2 FRAMEWORK FUNCTEST: APLICANDO PADRÕES DE SOFTWARE NA AUTOMAÇÃO DE TESTES FUNCIONAIS

Essa dissertação foi desenvolvida por Rafael Braga de Oliveira no ano de 2007, como requisito parcial para obtenção do Título de Mestre em Informática Aplicada, na Universidade de Fortaleza, demonstrando um framework para ampliação da reusabilidade e manutenebilidade de suítes de testes automatizados.

Devido os Scripts gerados a partir de ferramentas Record and Playback serem normalmente pouco reutilizáveis e pouco manuteníveis, e, à medida que aumenta a quantidade de casos de teste automatizados de uma suíte, torna-se cada vez mais difícil manter os scripts criados. De forma a minorar este problema, frameworks utilizando as técnicas *Data-driven* e *Keyword-driven* têm sido propostos para automatização de testes funcionais.

O trabalho desenvolvido pelo autor propõe um *framework* para ampliar a reusabilidade e a manutenebilidade de suítes de teste automatizadas. A solução foi desenvolvida no Serviço Federal de Processamento de Dados (SERPRO) e utilizada em projetos reais. O *framework*, denominado FuncTest, utiliza padrões de software e aplica as técnicas *Data-driven* e *Keyword-driven* na estruturação de suítes de teste automatizadas (OLIVEIRA, 2007).

Segundo o autor, um dos principais benefícios observados com o uso de testes automatizados foi a ampliação da cobertura dos testes durante a execução dos testes de regressão, que antes de se desenvolver o *framework*

possuíam escopo bastante restrito e tempo de execução significativamente superior.

8 O USO DAS TÉCNICAS DATA-DRIVEN E KEYWORD-DRIVEN NO DE DESENVOLVIMENTO DOS SCRIPTS DE AUTOMAÇÃO DE TESTES FUNCIONAIS DE SOFTWARE

O presente projeto refere-se ao desenvolvimento de um software que auxilie na criação e manutenção de scripts de automação de testes, para a execução de casos de testes funcionais automatizados em sistemas desktops, desenvolvidos em qualquer tipo de plataforma Windows.

Toda a estrutura do software baseia-se sempre na visão de dois usuários distintos, onde primeiramente existe o usuário responsável pela criação e manutenção dos scripts de automação de testes, ou seja, o usuário que detém o conhecimento técnico referente aos processos de automação de testes. Com uma visão um pouco diferente existe o usuário que irá criar os casos de testes com base em seu conhecimento da lógica de negócio do sistema a ser testado.

Nesse capítulo foram descritas as formas como o software proposto foi desenvolvido, bem como também as ferramentas utilizadas durante o processo de desenvolvimento do mesmo.

8.1 MODELAGEM DO SISTEMA

A modelagem de um sistema é o processo de desenvolvimento de modelos abstratos, onde cada modelo apresenta uma visão ou perspectiva diferente do sistema. Geralmente a modelagem representa algum tipo de notação gráfica, onde quase sempre é baseada em notação UML (linguagem de modelagem unificada, do inglês *Unified Modeling Language*) (SOMMERVILLE, 2003).

A seguir serão demonstradas algumas modelagens referentes ao software desenvolvido, com o objetivo de auxiliar no entendimento da estrutura do mesmo.

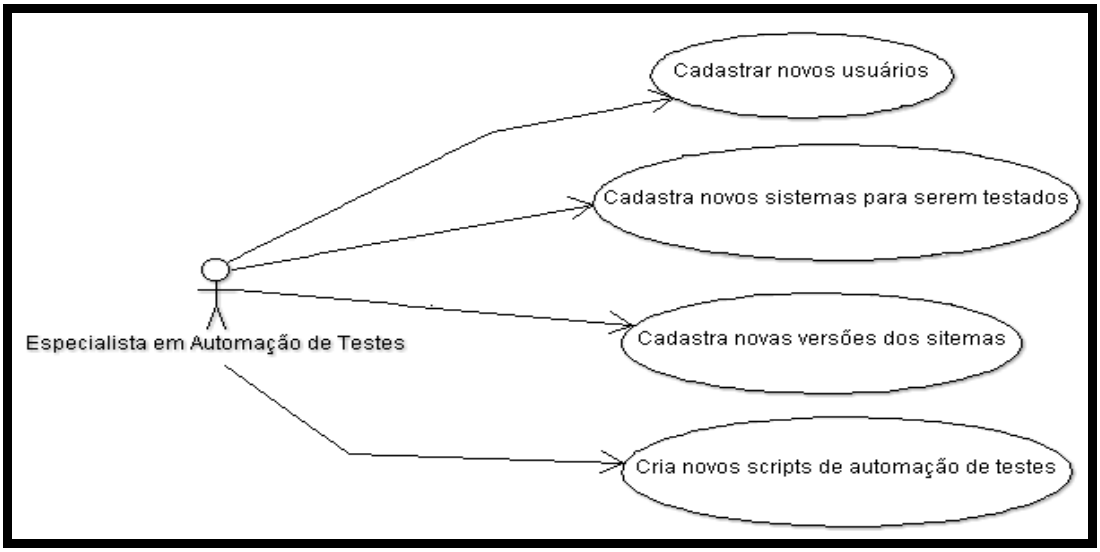
8.1.1 Diagrama de caso de uso

O diagrama de caso de uso procura demonstrar de forma gráfica o cenário que cada tipo de usuário espera do sistema.

Como já mencionado, existem dois perfis de usuários bem distintos no software desenvolvido. A seguir serão demonstrados os diagramas de caso de uso referente a cada perfil de usuário.

Na figura 7, é demonstrado um digrama de caso de uso referente aos usuários especialistas em automação de testes, conforme pode ser visto a seguir.

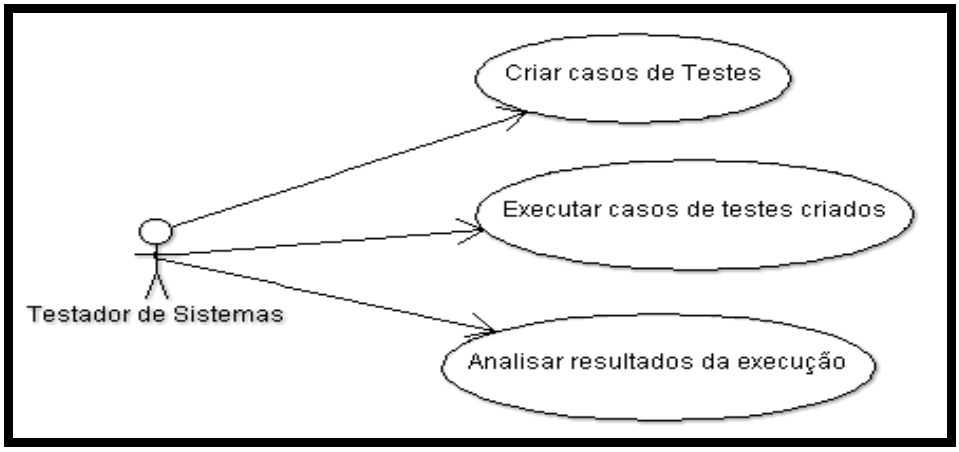
Figura 7 – Digrama de caso de uso do perfil de usuário Especialista em Automação de Testes.



Fonte: Do autor.

A seguir na figura 8, será demostrado um digrama de caso de uso referente ao perfil de usuário testador de sistemas, ou seja, os usuários que não detêm conhecimento técnico de automação de testes, mas sim a lógica de negócio do software a ser testado.

Figura 8 – Digrama de caso de uso do perfil de usuário Testador de Sistemas.



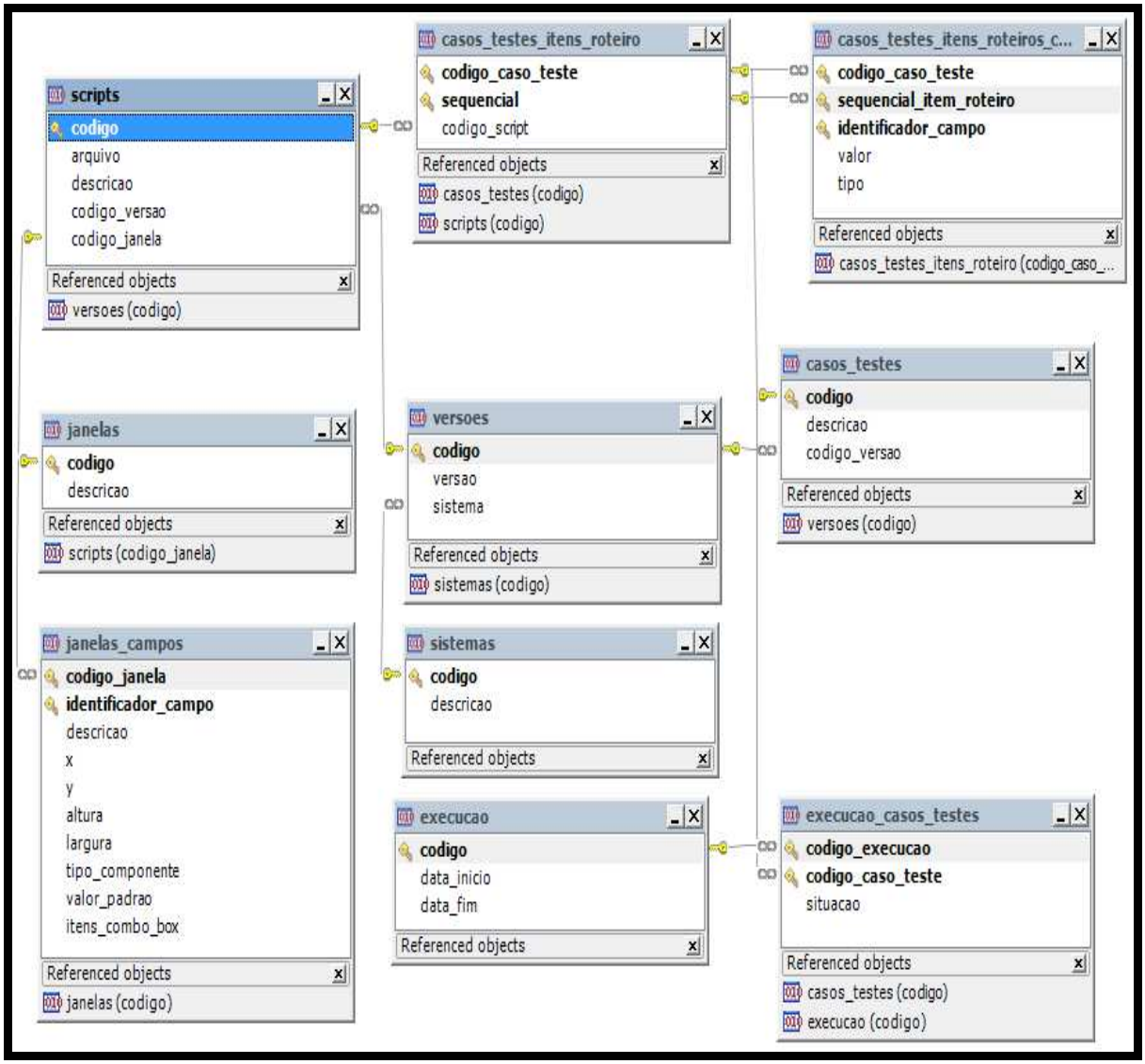
Fonte: Do autor.

8.1.2 Modelagem do Banco de Dados

A modelagem do banco de dados procura demonstrar as relações entre as entidades, suas especializações, atributos e auto-relações. Esse tipo de modelagem também tem como objetivo demonstrar as ligações entre as tabelas do banco de dados, as chaves primárias, os componentes de cada uma e etc.

A seguir na figura 9, é demonstrada a modelagem do banco de dados de utilizado no software desenvolvido.

Figura 9 – Modelagem do banco de dados utilizado pelo software desenvolvido.



Fonte: Do Autor.

8.2 FERRAMENTAS DE DESENVOLVIMENTO

Neste capitulo serão relatadas todas as ferramentas utilizadas no decorrer do desenvolvimento desse projeto.

8.2.1 JAVA

Foi utilizada a linguagem de programação JAVA, devido esta ser uma tecnologia que vem sendo muito utilizada no mercado devido a mesma ser

multi-plataforma, ou seja, possibilita que um software desenvolvido em um determinado sistema operacional seja executado também em outros sistemas operacionais.

Isso é possível pois diferentemente das linguagens convencionais que são compiladas para código nativo, a linguagem Java é compilada para um bytecode que é executado por uma máquina virtual.

Assim mesmo o software inicialmente ter sido desenvolvido e testado exclusivamente para na plataforma Microsoft Windows, o mesmo ficou disponível para ser estendido também para os demais sistemas operacionais do mercado.

Foi também utilizado o Netbeans versão 7.0.1 como IDE de desenvolvimento. O Netbeans é um ambiente integrado de desenvolvimento de softwares, que fornece os principais recursos para o desenvolvimento de aplicações em Java para desktop, web e para dispositivos móveis. Ele também é gratuito e de código aberto, possuindo uma grande comunidade de usuários e desenvolvedores por todo o mundo (NETBEANS, 2012, tradução nossa).

8.2.2 MySql

Como para o desenvolvimento do software foi necessário criar um pequeno banco de dados, foi utilizado o MySql como Sistema Gerenciador de Banco de Dados (SGBD).

Foi optado pelo MySql devido este possuir uma boa versão Freeware (grátis), ser de fácil uso, e também por possuir uma grande portabilidade, ou seja, possibilita que o software desenvolvido seja facilmente estendido para outros sistemas operacionais.

Segundo o próprio site do MySql, o mesmo pode ser executado em mais de 20 plataformas, incluindo Linux, Windows, Mac OS, Solaris, IBM AIX e entre outros, possibilitando um grande flexibilidade para o crescimento da aplicação nas mais diversas plataformas.

8.2.3 AutoIt3

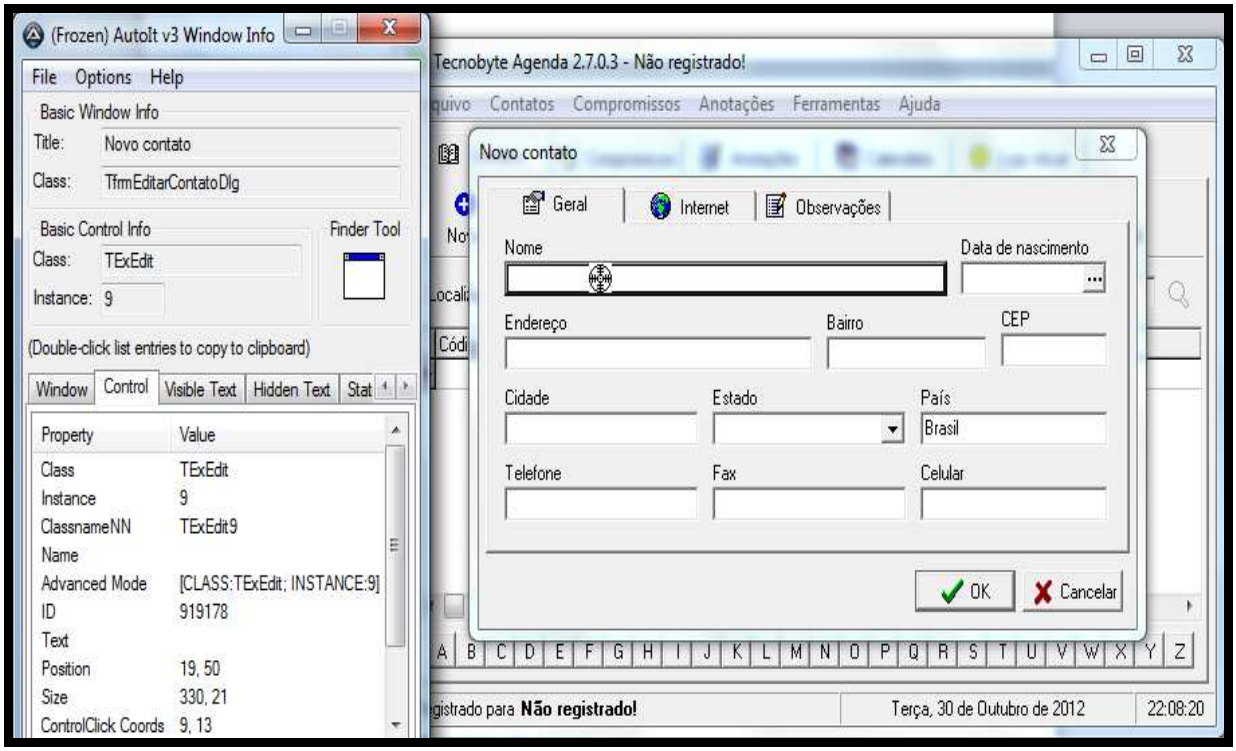
Como quase todas as ferramentas de automação de testes rodam com base em scripts de testes que interagem com a interface gráfica, foi utilizado o AutoIt em sua versão 3, que é uma linguagem de script freeware projetado para automatizar as ações dos usuário, tendo como base a interface gráfica. Esse usa a combinação de teclas, movimentos de mouse, e manipulação e controle de janelas, com o objetivo de automatizar as tarefas dos usuários (AUTOIT, 2012, tradução nossa).

8.2.3.1 Au3Info

Junto com o AutoIt3, vem uma ferramenta independente (executável) chamada de Au3Info. Esta permite que sejam obtidas várias informações das janelas especificas abertas no windows, tais como: título e tamanho de janelas, nome dos componentes (botões, inputTexts, checkBoxs e etc) visíveis e não visíveis das janelas (AUTOIT, 2012, tradução nossa).

Na figura 10 é demonstrada a fermenta em uso, onde a mesma está sendo utilizada para ser obter as informações do componente “*Nome*” da janela de cadastro de contatos do software Tecnobyte Agenda.

Figura 10 – Ferramenta Au3Info em uso.



Fonte: Do autor.

8.2.4 Tecnobyte Agenda

Para efeitos de testes do software desenvolvido, foi necessário utilizar algum software para que fossem criados os casos de testes que iriam testar o mesmo. Dessa forma foi pesquisado na Internet um software freeware de fácil uso e de interface gráfica simples, onde foi encontrado o software Tecnobytge Agenda versão 2.7.0.3.

O Tecnobyte Agenda pode ser encontrado através do site “www.tecnobyte.com.br”, onde se pode encontrar a sua versão freeware, como também as suas versões pagas.

8.3 O USO DA TÉCNICA DATA-DRIVEN

Conforme já explicado em tópicos anteriores, a técnica *Data-Driven* consiste em separar os scripts de testes da massa de dados que alimentam esses scripts, tornando o script de teste genérico para uma determinada quantidade de casos de testes.

A grande maioria das ferramentas de mercados estudadas no decorrer desse projeto implementam essa técnica através de planilhas de excel ou através de arquivos textos, onde existe um script que realiza a leitura desses arquivos e executam o caso de teste com base nas informações lidas.

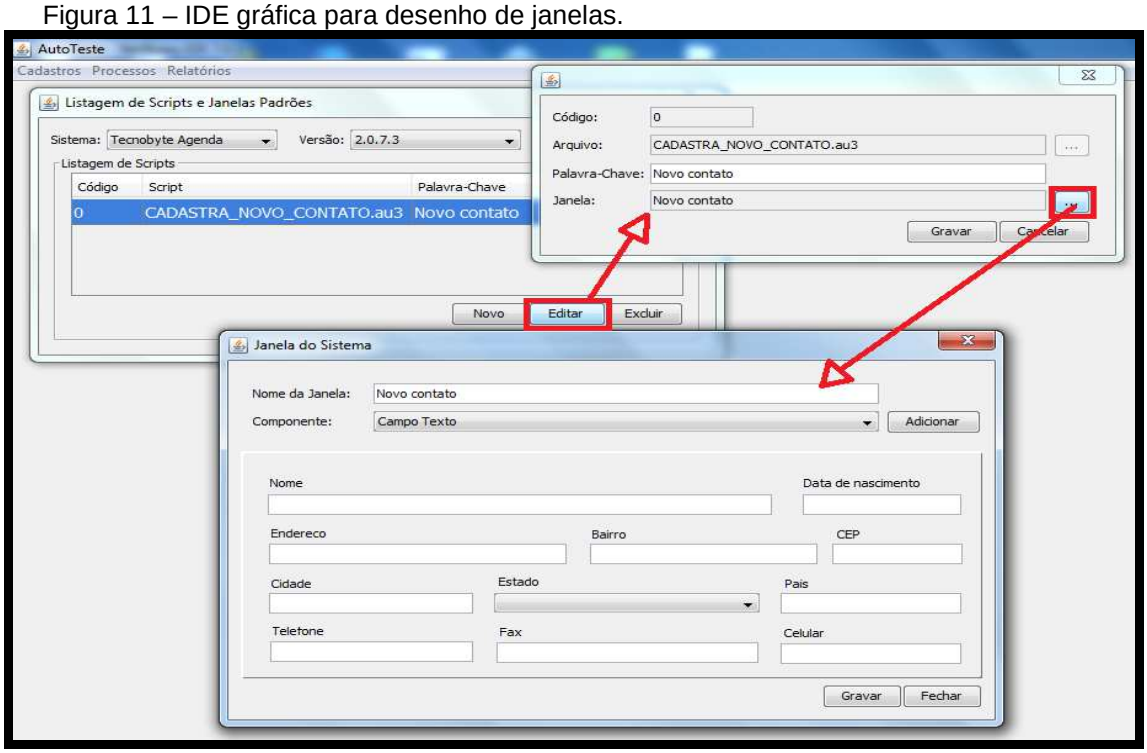
Pensando na manutenção da situação citada anteriormente se torna algo muito problemática, pois existem cenários em que para um mesmo script se deseja criar infinitas possibilidades de casos de testes, gerando assim uma grande quantidade de arquivos para serem gerenciados. Além disso, quando o usuário testador de sistemas está criando sua base de dados em planilhas, o mesmo não consegue ter uma noção exata de como é a respectiva janela que a planilha ou arquivo texto representa no seu sistema a ser testado.

Com base nessas informações, desenvolveu-se um software que tendo como principal objetivo auxiliar na manutenção e criação de casos de testes automatizados.

No que se refere a criação de scripts, foi utilizado o software AutoIt3 para realizar as iterações com o software a ser testado, simulando o usuário final. Já para fornecer os dados para esses scripts, foi desenvolvido no software uma espécie de IDE de desenvolvimento gráfico, onde através dessa o usuário consegue desenhar uma janela muito parecida com a do software do qual se deseja testar, vinculando essa janela ao script responsável em testar a mesma.

A figura 11 demonstra a janela onde se pode desenhar (simular) a respectiva janela do sistema a ser testado.

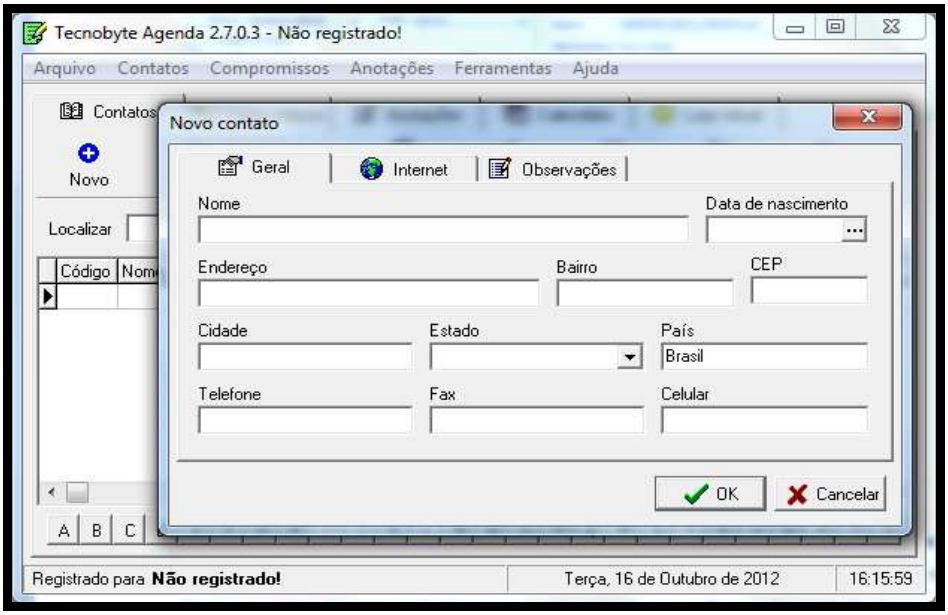
Figura 11 – IDE gráfica para desenho de janelas.



Fonte: Do autor.

A janela acima demonstrada é um exemplo de janela do software Tecnobyte Agenda, que já foi automatizada com o uso do software proposto. Na figura 12, é demonstrada a respectiva janela no software Tecnobyte Agenda.

Figura 12 – Janela de cadastro de contatos no software Tecnobyte Agenda.



Fonte: Do autor.

Como pode ser visto na figura 11, na janela existe um campo para que seja possível selecionar um script de teste. Para o exemplo da janela figura 12, foi criado o script demonstrado na figura 13, que será responsável mais a frente em preencher esta janela com as informações provenientes dos casos de testes.

Figura 13 – Exemplo de um script criado com o software AutoIt.

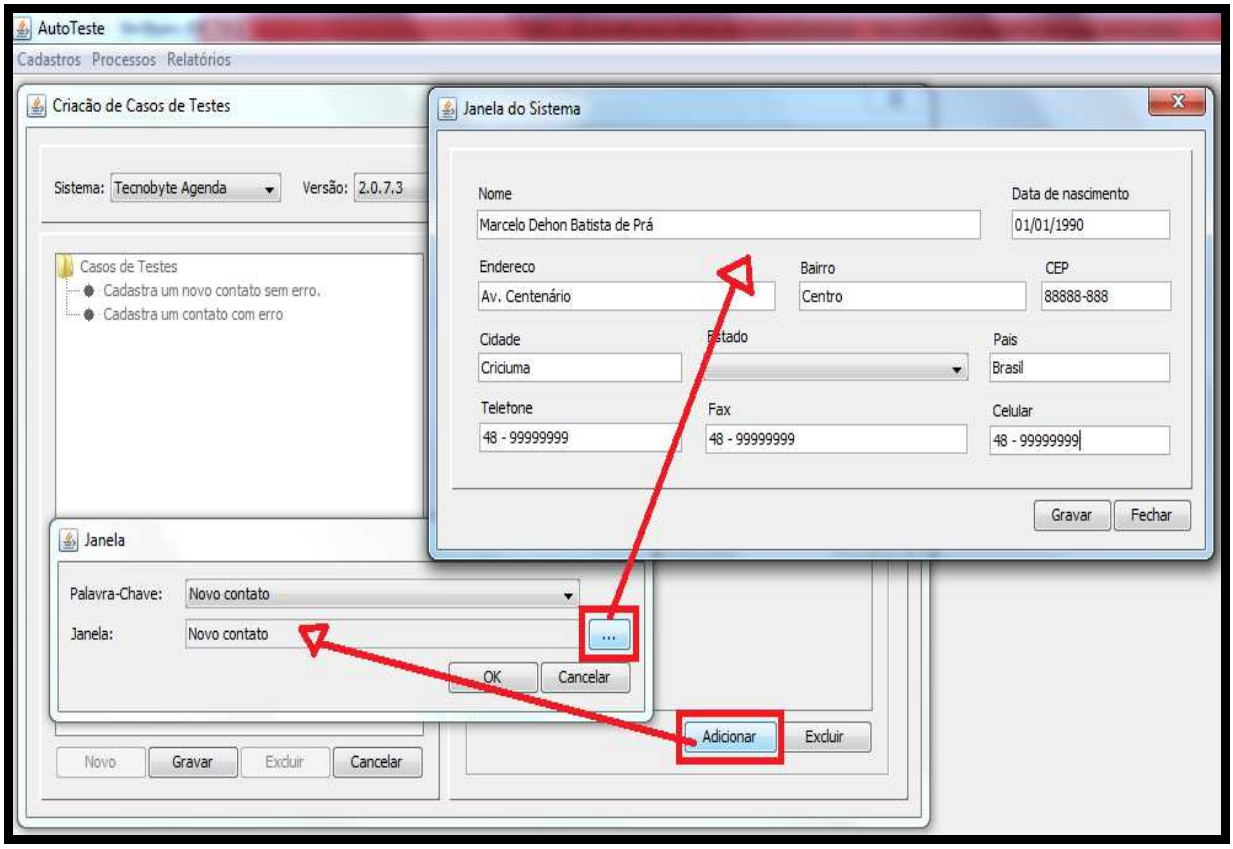
```
#cs ----------------------------------------------------------

 AutoIt Version: 3.3.6.1
 Author:         Marcelo Dehon Batista de Prá

 Script Function:
    Cadastra um novo contato.

#ce ----------------------------------------------------------

; Script Start - Add your code below here

#include "C:\AutoTeste\Scripts\GerenciaJanelas.au3"
#include "C:\AutoTeste\Scripts\PreencheCampos.au3"

aguardaJanelaTexto("Tecnobyte", "")

Send("{F2}")

aguardaJanela("Novo contato")

preencheCampos("Novo contato", "")

Send("{ENTER}")

aguardaJanelaTexto("Tecnobyte", "")
if(WinExists("Erro")) Then
    exit(5)
Else
    exit(0)
EndIf
```

Fonte: Do autor.

Depois de desenhada a janela e criado o script de teste, a mesma já estará disponível para que sejam criados novos casos de testes. Dessa forma, o usuário precisa selecionar no cadastro do caso de teste qual janela ele deseja automatizar e após preencher a mesma com os dados necessários. Feito isso basta gravar o caso de teste, e o mesmo já ficará disponível para ser executado quantas vezes forem necessárias.

A figura 14 mostra a criação de um novo caso de teste com base na janela já demonstrada na figura 10.

Figura 14 – Janela de criação de casos de testes, com a janela para alimentar o caso de teste aberta.



Fonte: Do autor.

Conforme demonstrado na figura 14, ao se clicar no botão *“Novo”* na janela de criação de casos de testes, é habilitado o botão *“Adicionar”*, para que seja possível selecionar as janelas, e informar os dados que serão preenchidos pelo script na respectiva janela original no software a ser testado.

8.3.1 Desenhando uma Janela

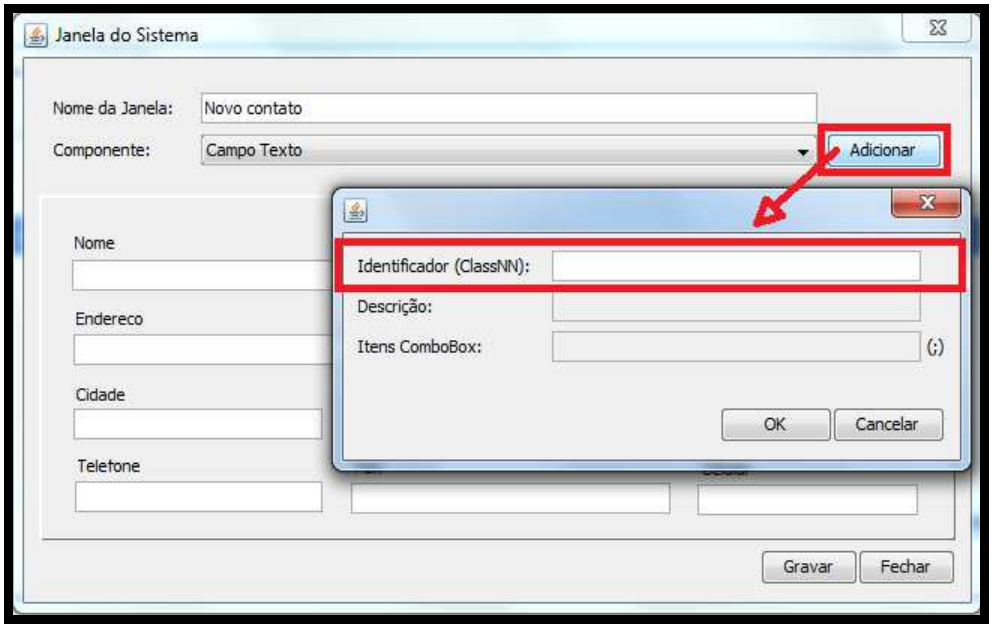
Conforme já explicado o software desenvolvido se baseia sempre em um script e em uma janela, onde os dados dessa janela são enviados para o script e este se encarrega de preencher a janela original do software a ser testado.

Conforme pode ser visto na figura 11, na janela de cadastro de scripts existe um botão [...] (reticências), onde ao clicar neste será possível criar uma janela com base em alguns componentes já prontos, no estilo Drag-and-Drop (arrastar e soltar).

A janela criada ficará vinculada ao script, dessa forma todos os dados preenchidos nesta janela no momento da criação dos casos de testes serão enviados para o script, e este se encarregará de executar aquilo ao qual foi programado.

Na figura 15, é demonstrada a janela aberta ao clicar-se no botão *“Adicionar”,* onde neste precisa-se informar as informações do componente que será adicionar a janela.

Figura 15 – Janela aberta ao clicar-se no botão “Adicionar”.



Fonte: Do autor.

Observe que na figura 15 existe uma campo chamado *"ClassNN"*, onde neste deve ser informado o ClassNN do referido componente na janela do sistema a ser testado. O ClassNN é um identificador baseado no nome da classe do componente mais o número da instância do mesmo, onde ambos podem ser determinado através de softwares espiões de janelas, onde no caso deste projeto foi usado o software Au3Info, conforme já explicado em tópicos anteriores.

O ClassNN deve ser informado exatamente o mesmo do componente, pois é através desse que o script criado através do AutoIt conseguirá identificador o componente na janela para preenchimento do mesmo.

8.4 O USO DA TÉNICA KEYWORD-DRIVEN

Conforme já explicado, a técnica *Keyword-Driven,* consiste em separar a criação dos scripts de testes, da criação dos casos de testes, deixando cada tarefa para o usuário que detêm mais conhecimento a respeito, ou seja, o usuário que detêm o conhecimento das técnicas e ferramentas de

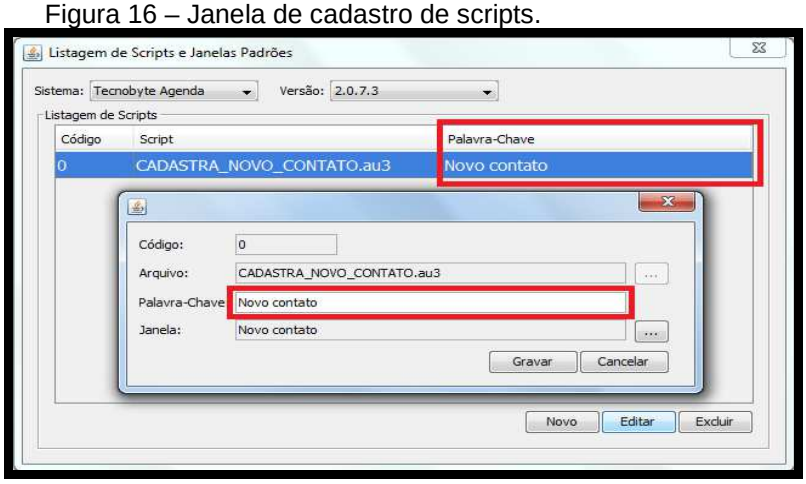
automação, fica responsável em criar os scripts de testes, enquanto isso, o usuário que detêm maior conhecimento da lógica de negócio do sistema a ser testado, fica com a tarefa de alimentar os scripts de testes criados pelo usuário especializado em automação.

Como o próprio nome da técnica já diz, essa é uma técnica a qual se baseia através de palavras chaves, ou seja, procura eliminar a complexidade dos scripts de testes, deixando esses mais específicos, com o principal objetivo de auxiliar o usuário no momento da criação dos casos de testes, onde basta o mesmo selecionar ou informar a palavra chave e passar as informações necessárias, para que o script de teste referente a esta palavra chave realize sua função ao qual este foi programado tendo como base de dados às informações passadas.

Dessa forma, no software desenvolvido foi criado um campo no cadastro de scripts denominado “Palavra-Chave”, para que o usuário informe uma palavra que identifique com poucas palavras o principal objetivo do script que está sendo cadastrado.

A figura 16, demonstra um script já cadastrado, onde a palavra chave deste é “Novo contato”, ou seja, a função desse script é fazer nada mais nada menos que apenas cadastrar um novo contato.

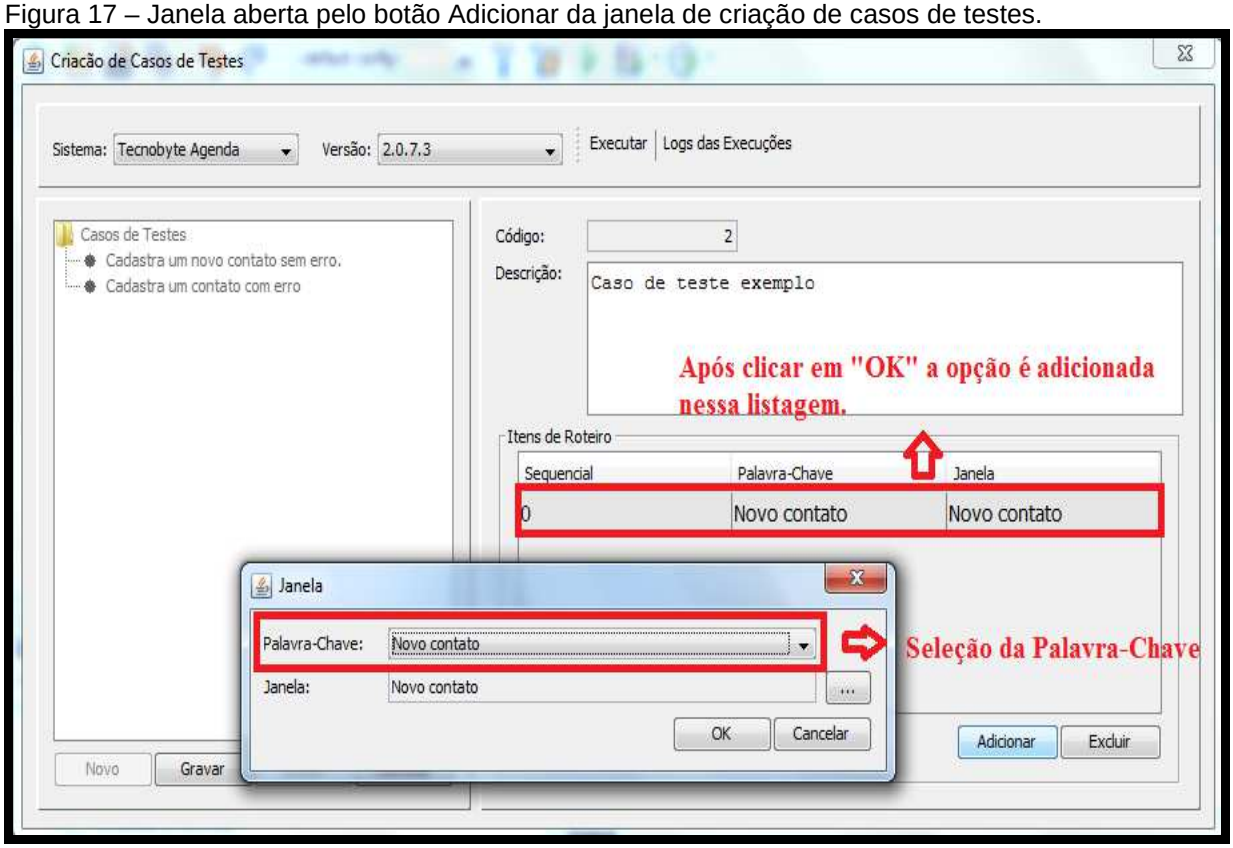
Figura 16 – Janela de cadastro de scripts.



Fonte: Do autor.

Depois de cadastrado o script e informada à palavra chave, o mesmo já ficará disponível para seleção na janela de criação de casos de testes, conforme pode ser visto na figura 17.

Figura 17 – Janela aberta pelo botão Adicionar da janela de criação de casos de testes.



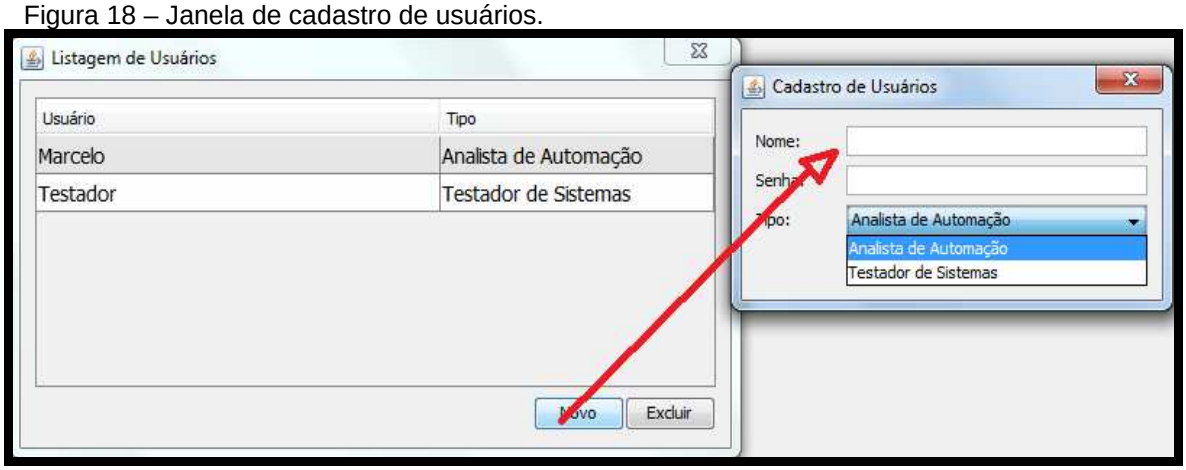
Fonte: Do autor.

8.4.1 Cadastro de Usuários

Para que a técnica *Keyword-Driven* pôde-se ser aplicada, se fez necessário a criação de um cadastro de usuários, para distinguir os usuários que desenvolvem as rotinas da automação, dos usuários testadores de sistemas que detêm o conhecimento técnico dos sistemas a serem testados.

Dessa forma através da opção "*Controle/Usuários*" da ferramenta, pode-se cadastrar dois tipos de usuários sendo eles: *"Analista de Automação"* e *"Testador de Sistemas",* conforme pode ser observado na figura 18.

Figura 18 – Janela de cadastro de usuários.



Fonte: Do autor.

Ao acessar a ferramenta será exibida uma janela para que o usuário possa fazer o seu login, dessa forma será identificado o usuário que está conectando e identificado o seu tipo, caso o tipo seja "*Analista de Automação*" será aberta a ferramenta para que possa ser usada da maneira completa, caso seja um usuário do tipo "*Testador de Sistemas*", será exibida unicamente a janela de criação de casos de testes, permitindo dessa forma somente cadastrar e excluir casos de testes.

8.5 MANUTENÇÃO DOS SCRIPTS DE TESTES EM VERSÕES DIFERENTES DO SISTEMA A SER TESTADO

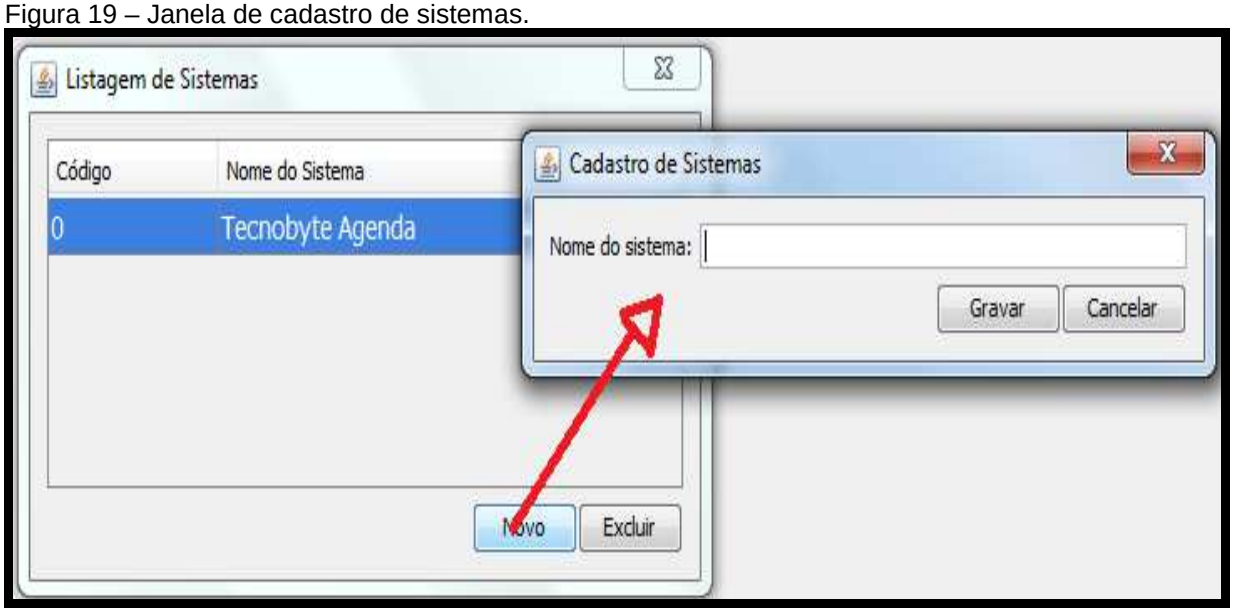
Sempre que uma janela ou rotina do software a ser testado é alterada exige uma pequena ou grande manutenção nos scripts e também nos casos de testes. Um dos grandes problemas que a maioria das ferramentas de mercado enfrentam é a manutenção acima citada.

Com base nisso, no software desenvolvido, foi implementado um cadastro de sistemas e versões dos softwares a serem testados, onde todos os scripts e casos de testes desenvolvidos irão ficar vinculados necessariamente a um sistema e versão.

Porém somente vincular o script e o caso de teste a um sistema e versão não resolve o problema da manutenção dos mesmos, com isso sempre

que uma versão for cadastrada é possível replicar todas as informações de uma versão anterior do mesmo sistema para a nova versão cadastrada. Dessa forma, podem-se realizar quaisquer alterações nos scripts e nos casos de testes em diversas versões que uma alteração não irá influenciar na outra. Abaixo na figura 19, é demostrada a janela de cadastro de sistemas.

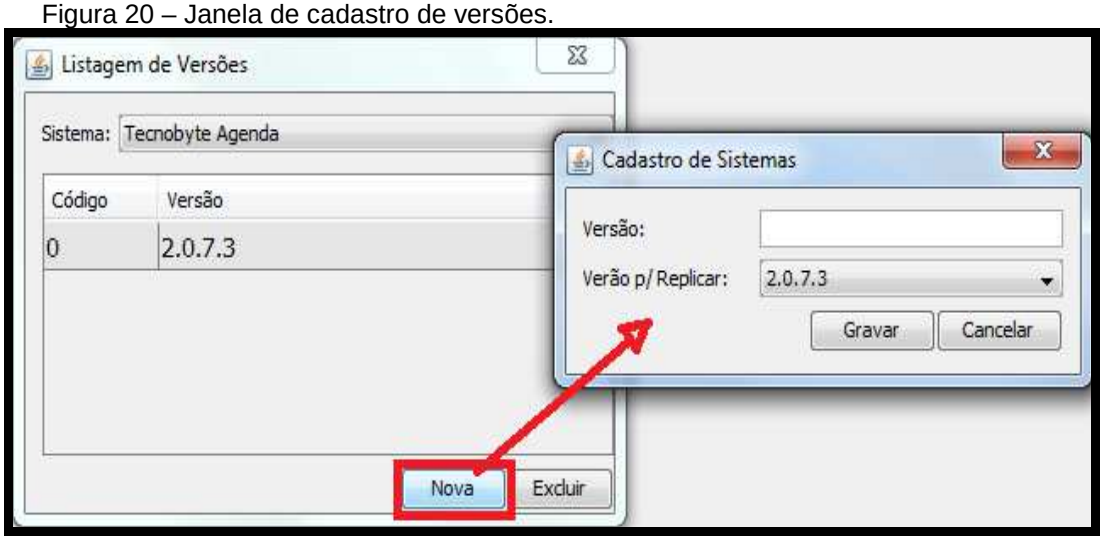
Figura 19 – Janela de cadastro de sistemas.



Fonte: Do autor.

Com o sistema cadastrado, pode-se cadastrar várias versões para esse sistema, conforme é demonstrado na figura 20.

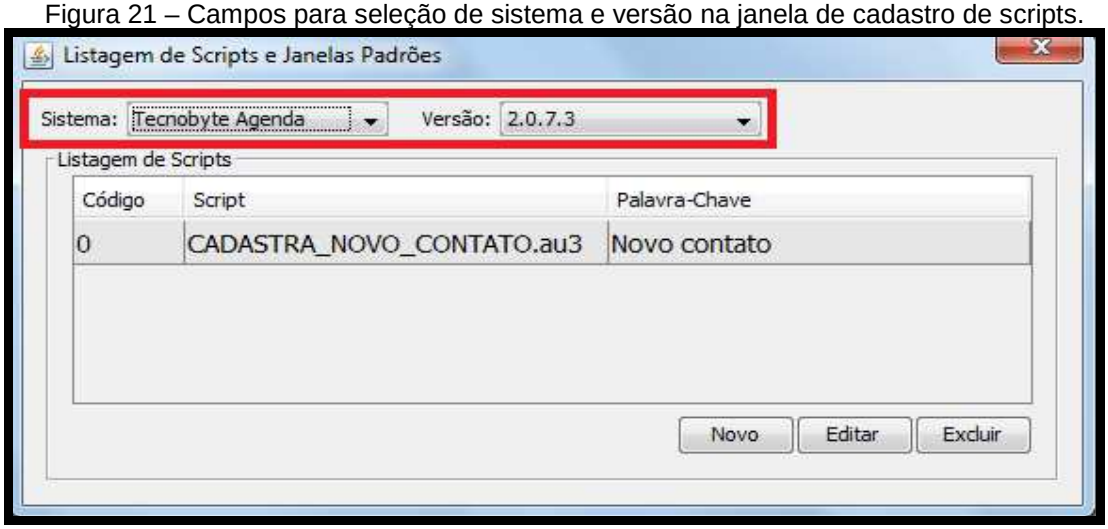
Figura 20 – Janela de cadastro de versões.



Fonte: Do autor.

Como pode ser observado nas figuras 21, quando cadastrado um *script*, sempre é necessário informar a qual versões esses estarão sendo vinculados.

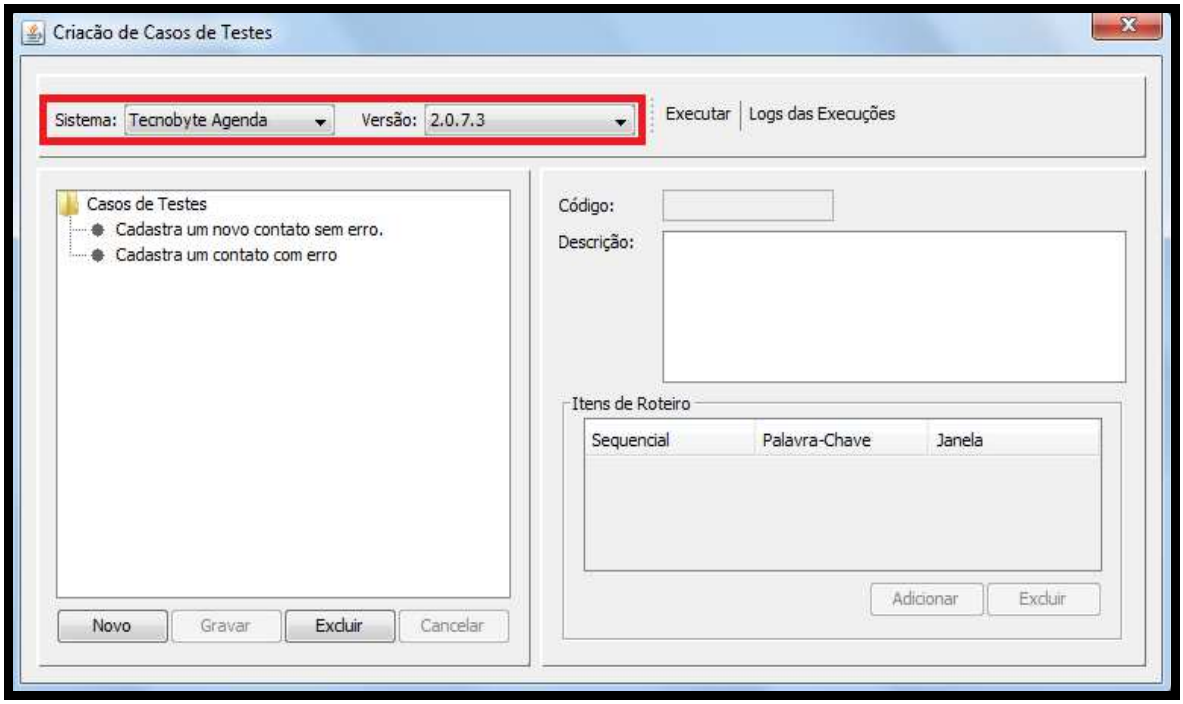
Figura 21 – Campos para seleção de sistema e versão na janela de cadastro de scripts.



Fonte: Do autor.

Igualmente ao cadatro de scripts, ao se cadastrar um caso de teste a seleção do sistema e versão aos quais este caso de teste será vinculado também é obrigatória, conforme pode ser observado na figura 22.

Figura 22 – Campos para seleção de sistema e versão na janela de cadastro de casos de testes.



Fonte: Do autor.

8.6 EXECUTANDO E VERIFICANDO O LOG

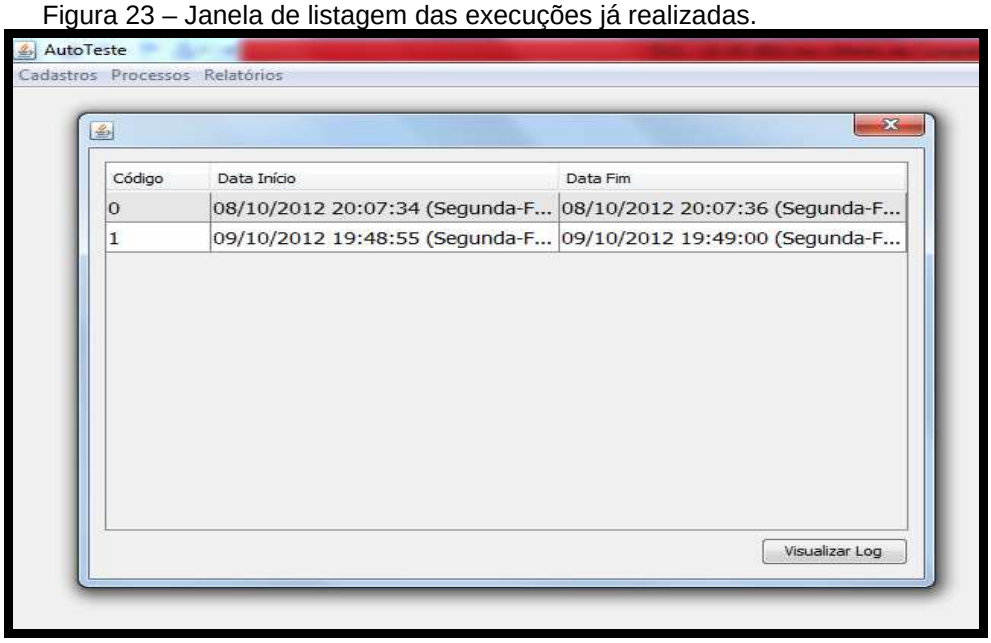
Depois de cadastrados todos os casos de testes, pode-se executar os mesmos tendo como objetivo final testar o software referente ao qual o caso de teste foi desenvolvido.

Durante a execução a ferramenta irá verificar se cada script de teste foi executado de forma correta, e irá salvando essa informação no banco de dados, para que ao término da execução seja montado um log para o usuário com o resultado da execução de cada caso de teste.

Ao término da execução o log ficará disponível através da opção *"Relatórios/Logs das Execuções"* ou através da opção *"Logs das Execuções"* da janela de criação de casos de testes.

Na figura 23 é demonstrada a janela contendo uma listagem com as execuções já realizadas. Para detalhar uma execução e visualizar o resultado de cada caso de teste, deve-se selecionar uma execução nessa listagem e após clicar em *"Visualizar log"*.

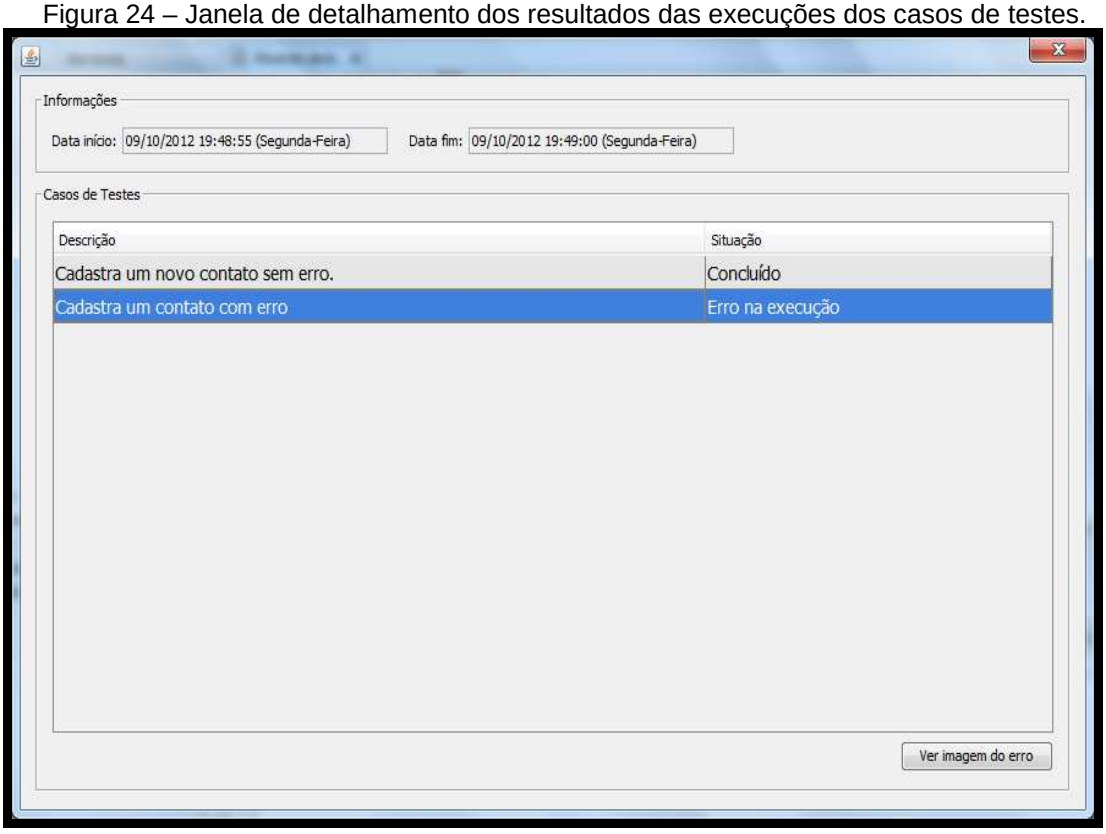
Figura 23 – Janela de listagem das execuções já realizadas.



Fonte: Do autor.

Selecionada a execução na listagem e clicado em *"Visualizar log"*, será demonstrada a janela da figura 24, onde terá um detalhamento da execução de cada caso de teste.

Figura 24 – Janela de detalhamento dos resultados das execuções dos casos de testes.



Fonte: Do autor.

Na figura 24, é demonstrado dois tipos de situações diferentes sendo elas *“Concluído”* e *“Erro na execução”*, sendo que para a situação *“Erro na execução”* estará habilitado o botão *“Ver imagem do erro”,* onde ao clicar no mesmo será demonstrado um ScreenShot tirado pela ferramenta no momento em que o referido erro do caso de teste ocorreu.

8.7 TESTES E RESULTADOS OBTIDOS

Conforme foi explicado no decorrer desse projeto, a execução dos testes é de grande importância para a qualidade do software, assim visando verificar se o software desenvolvido está de acordo com os requisitos inicias do projeto, foi criado um ambiente de testes e colocado em execução o software nesse ambiente.

Além de testar o software, um dos principais objetivos dos testes realizados, foi fazer uma comparação em relação ao tempo de execução dos testes automatizados se comparados com os testes manuais.

Conforme já mencionado anteriormente, o software utilizado nos testes foi o *“Tecnobyte Agenda”*, onde para efeito efeitos de testes foram criados 100 casos de testes referentes à janela de cadastro de “Contatos”.

Como não foram realizados os testes manuais dos 100 casos de testes, o tempo que se lavaria para executar os mesmo foi obtido através de estimativa, onde foi estimado um tempo aproximado de 1 hora.

Depois de cadastrados os 100 casos de testes na ferramenta, os mesmos foram colocados em execução, onde se obteve um tempo aproximado de 3 minutos.

Além dos 3 minutos para ter um embasamento mais preciso do tempo gasto com a automação dos 100 casos de testes, também foram levados em consideração os tempos gastos com planejamento, criação dos scripts, cadastros dos casos de testes, entre outros. Abaixo na tabela 2, é detalhado o tempo que foi utilizado em cada item.

Tabela 2 – Detalhamento dos tempos gastos com a automação de testes.

ITEM	TEMPO (MINUTOS)
Planejamento	30
Criação da janela	45
Criação do script	20
Criação dos 100 casos de testes	120
Execução dos 100 casos de testes	3
Total:	218 (3,6 horas)

Fonte: Do autor.

Assim chegou-se a conclusão que inicialmente até se desenvolver o *script*, a janela e criar os casos de testes, pode ser que o tempo de desenvolvimento seja maior do que se processo fosse testado manualmente. Mesmo assim, conforme já foi dito no decorrer desse projeto, o grande ganho da automação de testes está nos testes regressivos, onde após um caso de teste ser desenvolvido, este pode ser reexecutado N vezes em diversas versões do software a ser testado, onde a partir de uma determinada execução começará a ser mais lucrativa a automação de testes do que os testes manuais. Vale lembrar também que os testes manuais são muito mais sujeitos a falhas do que os automatizados.

9 CONCLUSÃO

Embora a automação de testes seja vista como um elemento importante para melhorar a produtividade no ambiente de testes, as atividades mais propicias para o uso de tal técnica são aquelas que envolvem a execução de tarefas repetitivas e exaustivas facilmente sucessíveis a erros humanos, ou difíceis de serem realizadas manualmente. Assim a automação de testes em hipótese alguma deve substituir os testes manuais, devendo esta ser introduzida como uma técnica adicional, cujo objetivo principal é aumentar a qualidade do produto a ser entregue.

Conforme já foi dito no decorrer de todo projeto, a automação de testes envolve um grande tempo de análise, planejamento, desenvolvimento e execução. Sendo que inicialmente se perde mais tempo criando do que se realizando os testes. Onde o grande potencial da automação se encontra nos testes regressivos, pois uma vez criado um caso de teste, esse ficará disponível para ser executado enumeras vezes. Ou seja, a automação de testes é um investimento que cujo retorno será em longo prazo.

Dessa forma esse projeto procurou criar um software que auxilia-se o processo de desenvolvimento e manutenção dos casos de testes automatizados, procurando diminuir o tempo de criação, planejamento e manutenção, tendo como base as técnicas *Data-Driven* e *Keyword-Driven*.

Como trabalhos futuros, pretende-se expandir o software desenvolvido também para outros sistemas operacionais, e também possibilitar a integração do software com componentes visuais mais complexos, possibilitando assim uma maior integração com os softwares disponíveis atualmente no mercado.

REFERÊNCIAS

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

CHEQUE, Paulo. **A Importância dos Testes Automatizados,** São Paulo [2008]. Disponível em: <http://www.ime.usp.br/~kon/papers/EngSoftMagazineIntroducaoTestes.pdf> Acessado em: 24 out. 2011.

CRAIG, Rick D., JASKIEL, Stefan P. ***Systematic Software Testing***. United States of America: Artech House, 2002.

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

DUSTIN, Elfriede. ***Automated Software Testing: Introduction, Management, and Performance***. Bk&CD Rom edition, 1999.

FEWSTER, Mark; GRAHAM, Dorothy. ***Software Test Automation: effective use of test execution tools.*** Inglaterra, Londres: Addison-Wesley, 1999.

HAYES, Linda. ***Evolution of Automated Software Testing***. Automated Software Testing Magazine. Brasil, p.14-20. Agosto 2009.

KOSCIANSKI, André; SOARES, Michel Dos Santos. ***Qualidade de Software:*** Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. São Paulo: Novatec, 2006.

LAGES, Daniel Scaldaferri. ***Automação de Testes:*** um logo na pele de cordeiro?. Engenharia de software magazine. Brasil, p.20-25. 2011.

LOPES, Fernando Bettiol. ***Desenvolvimento de uma Metodologia Aplicada ao Gerenciamento e Acompanhamento de Teste de Software via Web.*** 2009. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - UNESC. Criciúma.

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

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

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

OLIVEIRA, Rafael Braga de. ***Framework functest: aplicando padrões de software na automação de testes funcionais.*** 2007. Dissertação (Mestrado em Informática Aplicada) - UNIFOR. Fortaleza.

PAULA FILHO, Wilson de Pádua. ***Engenharia de Software:*** Fundamentos, Métodos e Padrões. Rio de Janeiro: LTC, 2003.

PFLEEGER, Shari Lawrence. ***Engenharia de Software: Teoria e Prática.*** São Paulo: Pearson Prentice Hall, 2004.

PRESSMAN, Roger S.. ***Engenharia de software****.* São Paulo: Person, 1995.

RAMIREZ, Jaime Arturo. **Teste de software**, Belo Horizonte, dez. [1990]. Disponível em: <http://ead1.eee.ufmg.br/~renato/engsoft/Teste_Soft.pdf>. Acesso em: 15 out. 2011.

SCALDAFERRI, Daniel. ***Automação dos Testes: um lobo na pele de cordeiro*****.** Revista Engenharia de Software magazine, Edição 29, Ano 3 p. 20-25, 2010.

SOMMERVILLE, Ian. ***Engenharia de Software.*** São Paulo: Pearson, 2003.

SCALDAFERRI, Daniel. **Automação dos Testes: um lobo na pele de cordeiro.** Revista Engenharia de Software magazine, Edição 29, Ano 3 p. 20-25, 2010.

NETBEANS. Site oficial da IDE NetBeans. Disponível em: <http://www.netbeans.org>. Acesso em: 23 out. 2012.

MYSQL. Site oficial do SGDB MySql. Disponível em: <http://www.mysql.com>. Acesso em: 23 out. 2012.

AutoIt. Site oficial do Autoit3. Disponível em: < http://www.autoitscript.com/site >. Acesso em: 30 out. 2012.

APÊNDICE(S)

APÊNDICE A – Artigo Científico

O Uso das Técnicas Data-Driven e Keyword-Driven no Processo de Desenvolvimento dos Scripts de Automação de Testes Funcionais de Software

Marcelo D. B. de Prá[1], Ana Claudia G. Barbosa[2]

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

[2]Professora Orientadora do Curso de Ciência da Computação – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma – SC

marcelo.dehon@hotmail.com, agb@unesc.net

Abstract. *This project seeks to demonstrate some key techniques for test automation, and develop a tool that assists in the creation and execution of automated test cases, based on the technical Data-Driven and Keyword-Driven for the creation of test scripts execution. Numerous studies emphasize the importance of an effective testing process to minimize the cost and provide higher quality in software development. The most effective testing processes before starting any program be written. In this context, tests can be designed from step requirements. The manual execution of a test is quick and effective, but the execution and repetition of a wide range of tests manually is time consuming and tiring. The test automation increases productivity and achieves in a shorter time than what is generally repetitive in the test environment.*

Resumo. *Esse projeto procura demonstrar algumas das principais técnicas de automação de testes, e desenvolver uma ferramenta que auxilie na criação e execução de casos de testes automatizados, baseando-se nas técnicas Data-Driven e Keyword-Driven para a criação dos scripts de execução dos testes. Inúmeros estudos ressaltam a importância de um processo de teste efetivo para minimizar o custo e proporcionar maior qualidade no desenvolvimento de software. Os processos de teste mais efetivos iniciam antes de qualquer programa ser escrito. Neste contexto, testes podem ser planejados desde a etapa de requisitos. A execução manual de um teste é rápida e efetiva, mas a execução e repetição de um vasto conjunto de testes manualmente é uma tarefa demorada e cansativa. A automação de testes aumenta a produtividade e atinge em um tempo menor aquilo que em geral é repetitivo no ambiente de testes.*

1. Introdução

Automatizar um teste significa utilizar algum software que imita a interação com a aplicação no que se refere ao teste tal qual um ser humano faria (com algumas limitações). A principal razão do uso e da disseminação da automação dos testes de software é justamente a urgência cada vez maior de realizar mais testes em menos tempo. As aplicações ficaram mais complexas ao mesmo tempo em que se passou a exigir maior qualidade do produto entregue (MOLINARI, 2010).

A automação de testes funcionais visa automatizar um ou mais casos de testes funcionais. Testes funcionais visam testar as principais funcionalidades de um sistema, no qual se destacam os testes de interface (MOLINARI, 2010).

Muitas empresas ao tentar implantar um processo de automação de testes, seja ela funcional ou não (testes de desempenho, caixa branca e etc...) dentro de seu ambiente de produção, fracassam nas fases iniciais durante o processo de implantação, por alguns motivos como: expectativas irreais que ocorrem geralmente quando os gestores e testadores deixam-se seduzir pelas promessas dos fabricantes de software de automação, onde quando começam a utilizar as ferramentas percebem que nem sempre é tão fácil automatizar como prometia os fabricantes, porém não é algo inatingível. Também por não tratarem a automação de testes como um novo projeto. Onde muitas vezes ocorre dos gestores acreditarem erroneamente que um processo de teste manual, e um processo automatizado são as mesmas coisas.

Dessa forma, esse projeto de pesquisa demonstra algumas das principais técnicas de automação de testes, e tem como objetivo principal desenvolver uma ferramenta que auxilie na criação e execução de casos de testes automatizados, baseando-se nas técnicas *Data-Driven e Keyword-Driven* para a criação dos scripts de execução dos testes.

2. Automação de Testes Funcionais

Pode-se diferenciar a tarefa de teste de automação de teste da seguinte maneira. Na primeira, é realizada a tarefa de testar, e na segunda é a utilização de um software imitando um ser humano no que diz respeito à integração com a aplicação (FEWSTER; GRAHAM, 1999, tradução nossa).

A automação de testes aumenta em muito a produtividade e atinge em um tempo muito menor aquilo que em geral é repetitivo no ambiente de teste. Contudo os testes manuais não podem ser eliminados em hipótese alguma, sendo que os mesmos devem ser focados naquilo que é muito caro de se automatizar (MOLINARI, 2010).

Sendo assim, pode-se afirmar que o grande potencial da automação de testes funcionais, está nos testes de regressão. Pois basta automatizar uma vez um determinado teste, para que o mesmo possa ser repetido quantas vezes forem necessárias (MOLINARI, 2010).

2. Técnicas para Criação e Manutenção de Scripts de Automação de Testes

Segundo Graham e Fewster (1999, tradução nossa), existem cinco grandes técnicas para projetar e desenvolver scripts para as ferramentas de automação de testes:

f) scripts linerares;

g) scripts estruturados;

h) scripts compartilhados;

i) data-driven scripts;

j) keyword-driven scripts.

As técnicas acimas não exclusivas, ou seja, não devem ser utilizadas individualmente e sim em conjunto, onde cada um tem suas vantagens e desvantagens.

2.1. Scripts Lineares

k) Os scripts lineares são aqueles que implementam suas instruções de forma sequencial. Em muitos casos de testes estes scripts são criados através de ferramentas do tipo Record & Playback, as quais somente implementam a criação de scripts lineares (FEWSTER; GRAHAM, 1999, tradução nossa).

Geralmente esse tipo de script é simples e fácil de ser desenvolvido, porém por outro lado são muitos vulneráveis à mudança do software, tendo assim uma manutenção mais cara (MOLINARI, 2010)

2.2. Scripts Estruturados

l) Scripts estruturados são scripts que implementam estruturas de controle especiais (seleção ou interação) ou estrutura de chamada a outros scripts, portanto, estes scripts são programados, mesmo que parcialmente (FEWSTER; GRAHAM, 1999, tradução nossa).

m) A estrutura seleção avalia condições que determinam se um conjunto específico de instruções será executado. A estrutura interação possibilita repetir um conjunto de instruções uma determinada quantidade de vezes. Esta estrutura é utilizada, por exemplo, para a leitura de uma sequência de registros num arquivo de dados. Neste caso, a sequência pode ser executada até que seja concluída a leitura de todos os registros (FEWSTER; GRAHAM, 1999, tradução nossa).

Além disso, um script pode chamar outro *script*. Este técnica pode ser usada para dividir scripts grandes em scripts menores e mais gerenciáveis, melhorando o seu reuso (FEWSTER; GRAHAM, 1999, tradução nossa).

2.3. Scripts Compartilhados

Scripts compartilhados são scripts utilizados por mais de um caso de teste. O uso desta técnica visa identificar tarefas repetitivas que possam ser reutilizadas por mais de um caso de teste. Neste caso, são criados scripts que realizam tais tarefas, para que sejam chamados num ponto específico de um caso de teste (FEWSTER; GRAHAM, 1999, tradução nossa).

O compartilhamento destes scripts pode ser feito entre casos de teste de um mesmo sistema ou de diferentes sistemas. Entre as vantagens desta técnica, destaca-se principalmente a melhoria do reuso de *scripts*. Entre as desvantagens, destaca-se a existência de um número maior de scripts para serem mantidos (FEWSTER; GRAHAM, 1999, tradução nossa).

2.4. Data-Driven Scripts

n) A técnica data-driven (dirigido a dados) consiste em extrair dos scripts de teste os dados de teste, que são específicos por caso de teste, e armazená-los em arquivos separados dos scripts de teste. Os scripts de teste passam a conter apenas os procedimentos de teste (lógica de execução) e as ações de teste sobre a aplicação, que normalmente são genéricos para um conjunto de casos de teste. Assim, os scripts de teste não mantêm os dados de teste no próprio código, obtendo-os diretamente de um arquivo separado, somente quando necessário e de acordo com o procedimento de teste implementado (FEWSTER; GRAHAM, 1999, tradução nossa).

2.5. Keyword-Driven Scripts

o) A técnica *Keyword-driven* (dirigido a palavras chaves) propõe separar o desenvolvimento dos casos de teste do desenvolvimento dos *scripts* de teste. Isto possibilita que testadores experientes no domínio da aplicação concentrem-se na criação dos arquivos de teste, enquanto que profissionais com conhecimento técnico concentrem-se nos scripts de suporte (FEWSTER; GRAHAM, 1999, tradução nossa).

p) O uso da técnica *Keyword-driven* diminui a complexidade dos *scripts*, pois, assim como ocorre quando descrevemos um teste manual, permite que os casos de teste sejam especificados de maneira menos detalhada (FEWSTER; GRAHAM, 1999, tradução nossa).

3. O Uso da Técnica Data-Driven

A técnica *Data-Driven* consiste em separar os scripts de testes da massa de dados que alimentam esses scripts, tornando o script de teste genérico para uma determinada quantidade de casos de testes.

No que se refere a criação de scripts, foi utilizado o software AutoIt3 para realizar as iterações com o software a ser testado, simulando o usuário final. Já para fornecer os dados para esses scripts, foi desenvolvido no software uma espécie de IDE de desenvolvimento gráfico, onde através dessa o usuário consegue desenhar uma janela muito parecida com a do software do qual se deseja testar, vinculando essa janela ao script responsável em testar a mesma.

A figura 1 demonstra a janela onde se pode desenhar (simular) a respectiva janela do sistema a ser testado.

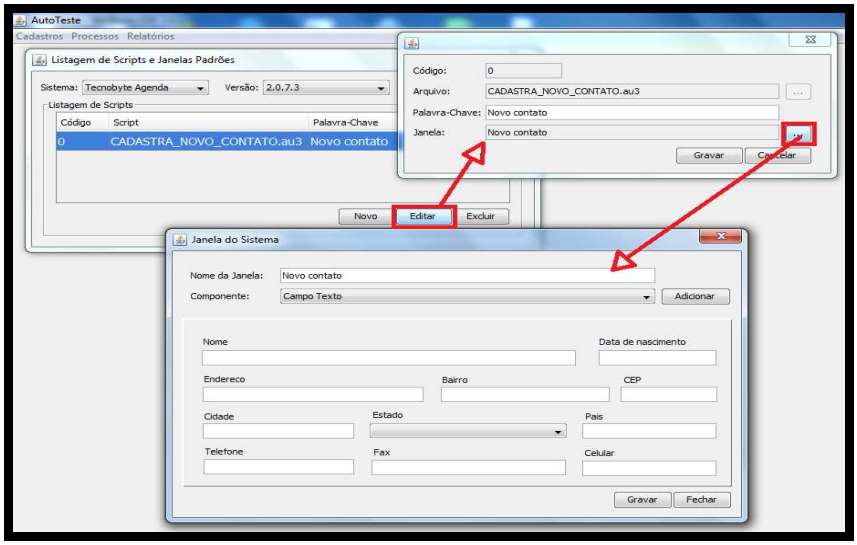


Figura 1 - IDE gráfica para desenho de janelas.

A janela acima demonstrada é um exemplo de janela do software Tecnobyte Agenda, que já foi automatizada com o uso do software proposto. Na figura 2, é demonstrada a respectiva janela no software Tecnobyte Agenda.

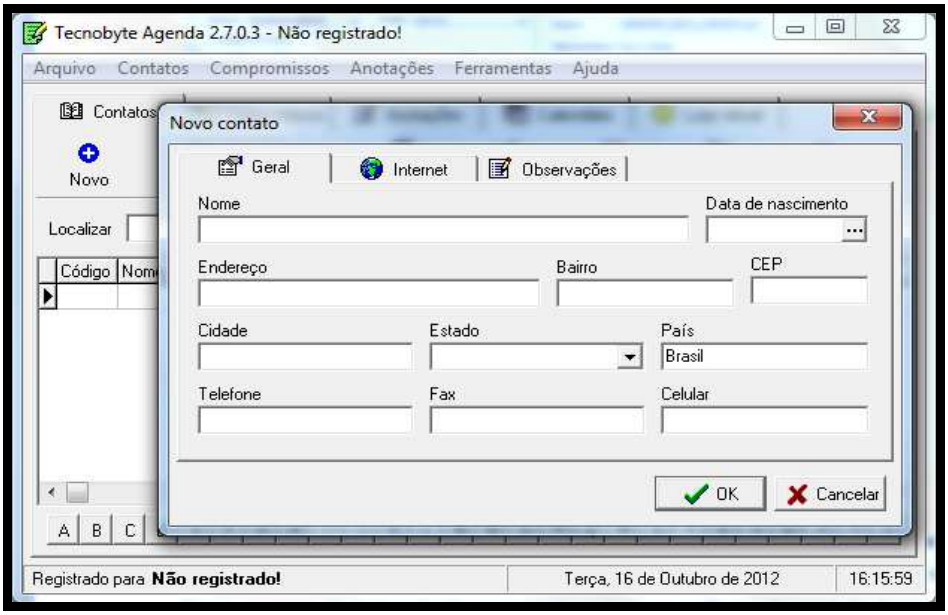


Figura 2 - Janela de cadastro de contatos no software Tecnobyte Agenda.

Como pode ser visto na figura 1, na janela existe um campo para que seja possível selecionar um script de teste.

Depois de desenhada a janela e criado o script de teste, a mesma já estará disponível para que sejam criados novos casos de testes. Dessa forma, o usuário precisa selecionar no cadastro do caso de teste qual janela ele deseja automatizar e após preencher a mesma com os dados necessários. Feito isso basta gravar o caso de teste, e o mesmo já ficará disponível para ser executado quantas vezes forem necessárias.

A figura 3 mostra a criação de um novo caso de teste com base na janela já demonstrada na figura 1.

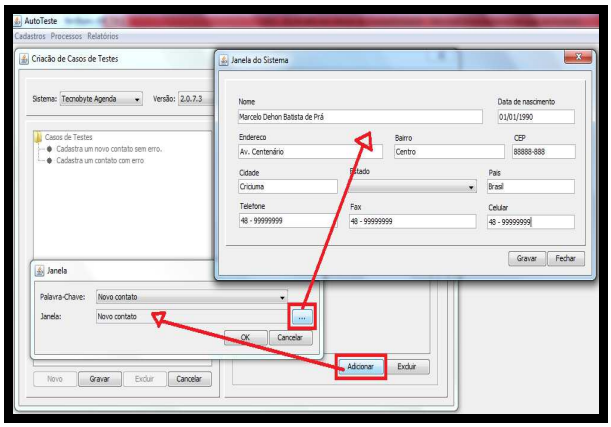


Figura 3 - Janela de criação de casos de testes, com a janela para alimentar o caso de teste aberta.

Conforme demonstrado na figura 3, ao se clicar no botão *"Novo"* na janela de criação de casos de testes, é habilitado o botão *"Adicionar"*, para que seja possível selecionar as janelas, e informar os dados que serão preenchidos pelo script na respectiva janela original no software a ser testado.

4. O Uso da Técnica Keyword-Driven

A técnica *Keyword-Driven*, consiste em separar a criação dos scripts de testes, da criação dos casos de testes, deixando cada tarefa para o usuário que detêm mais conhecimento a respeito, ou seja, o usuário que detêm o conhecimento das técnicas e ferramentas de automação, fica responsável em criar os scripts de testes, enquanto isso, o usuário que detêm maior conhecimento da lógica de negócio do sistema a ser testado, fica com a tarefa de alimentar os scripts de testes criados pelo usuário especializado em automação.

Como o próprio nome da técnica já diz, essa é uma técnica a qual se baseia através de palavras chaves, ou seja, procura eliminar a complexidade dos scripts de testes, deixando esses mais específicos, com o principal objetivo de auxiliar o usuário no momento da criação dos casos de testes, onde basta o mesmo selecionar ou informar a palavra chave e passar as informações necessárias, para que o script de teste referente a esta palavra chave realize sua função ao qual este foi programado tendo como base de dados às informações passadas.

Dessa forma, no software desenvolvido foi criado um campo no cadastro de scripts denominado "Palavra-Chave", para que o usuário informe uma palavra que identifique com poucas palavras o principal objetivo do script que está sendo cadastrado.

A figura 4, demonstra um script já cadastrado, onde a palavra chave deste é "Novo contato", ou seja, a função desse script é fazer nada mais nada menos que apenas cadastrar um novo contato.

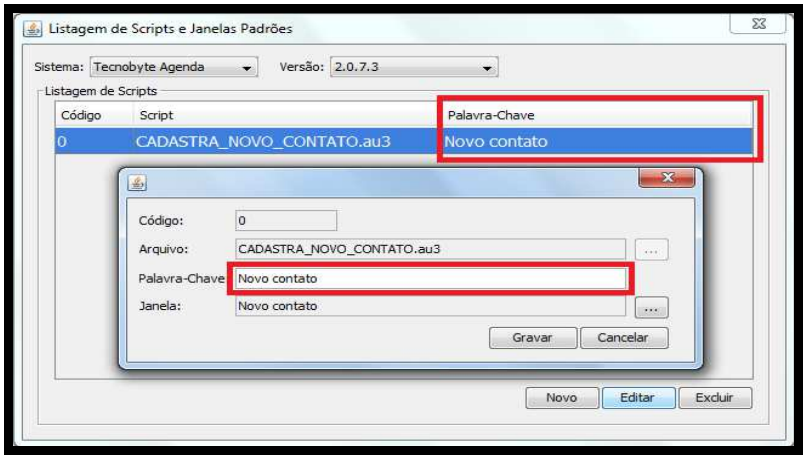


Figura 4 - Janela de cadastro de scripts.

Depois de cadastrado o script e informada à palavra chave, o mesmo já ficará disponível para seleção na janela de criação de casos de testes, conforme pode ser visto na figura 5.

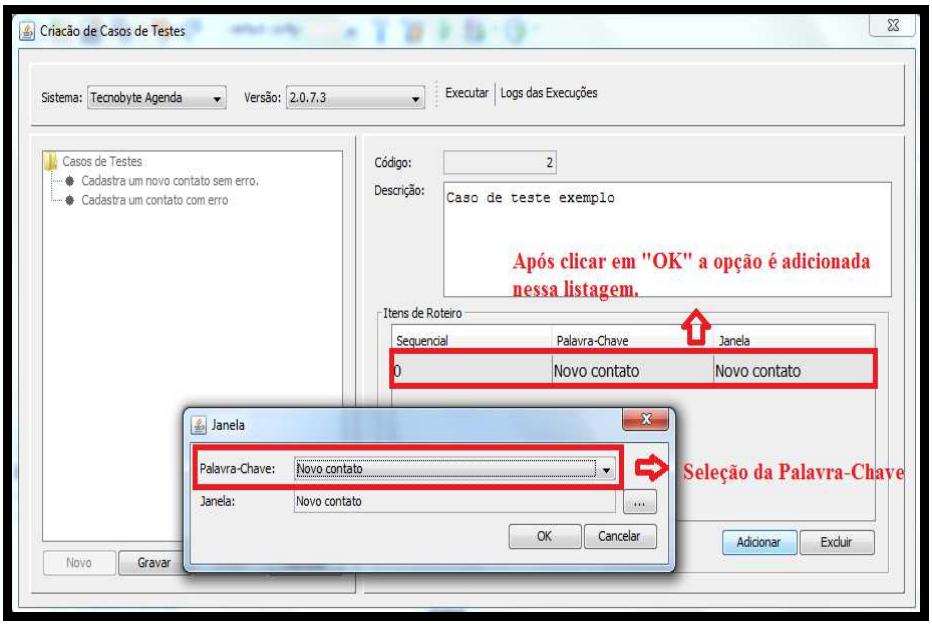


Figura 5 - Janela aberta pelo botão Adicionar da janela de criação de casos de testes.

5. Executando e Verificando o Log

Depois de cadastrados todos os casos de testes, pode-se executar os mesmos tendo como objetivo final testar o software referente ao qual o caso de teste foi desenvolvido.

Durante a execução a ferramenta irá verificar se cada script de teste foi executado de forma correta, e irá salvando essa informação no banco de dados, para que ao término da execução seja montado um log para o usuário com o resultado da execução de cada caso de teste.

Ao término da execução o log ficará disponível através da opção *"Relatórios/Logs das Execuções"* ou através da opção *"Logs das Execuções"* da janela de criação de casos de testes.

Na figura 6 é demonstrada a janela contendo uma listagem com as execuções já realizadas. Para detalhar uma execução e visualizar o resultado de cada caso de teste, deve-se selecionar uma execução nessa listagem e após clicar em *"Visualizar log"*.

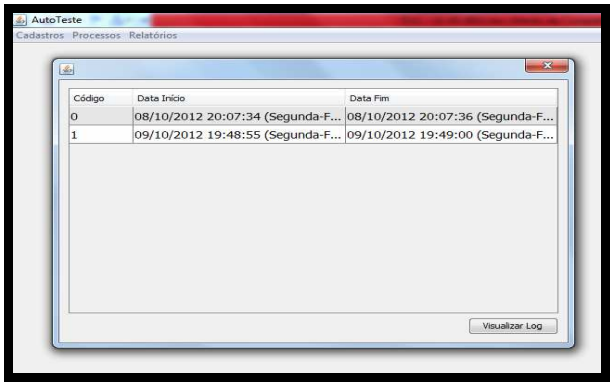


Figura 6 – Janela de listagem das execuções já realizadas

Selecionada a execução na listagem e clicado em *"Visualizar log"*, será demonstrada a janela da figura 7, onde terá um detalhamento da execução de cada caso de teste.

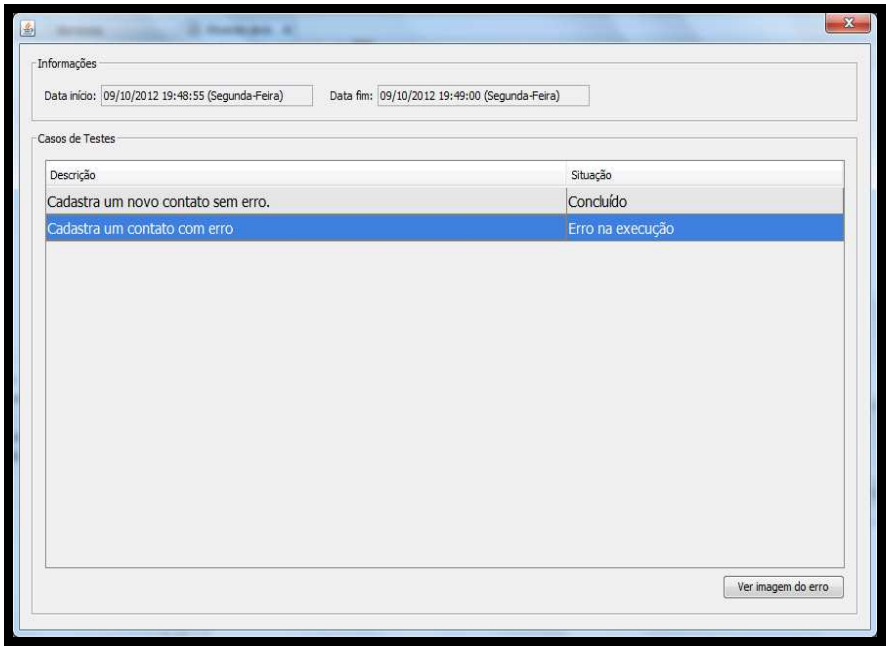


Figura 7 - Janela de detalhamento dos resultados das execuções dos casos de testes.

Na figura 7, é demonstrado dois tipos de situações diferentes sendo elas *"Concluído"* e *"Erro na execução"*, sendo que para a situação *"Erro na execução"* estará habilitado o botão *"Ver imagem do erro"*, onde ao clicar no mesmo será demonstrado um ScreenShot tirado pela ferramenta no momento em que o referido erro do caso de teste ocorreu.

6. Resultados Obtidos

Conforme foi explicado no decorrer do projeto, a execução dos testes é de grande importância para a qualidade do software, assim visando verificar se o software desenvolvido está de acordo com os requisitos inicias do projeto, foi criado um ambiente de testes e colocado em execução o software nesse ambiente.

Além de testar o software, um dos principais objetivos dos testes realizados, foi fazer uma comparação em relação ao tempo de execução dos testes automatizados se comparados com os testes manuais.

Conforme já mencionado anteriormente, o software utilizado nos testes foi o *"Tecnobyte Agenda"*, onde para efeito efeitos de testes foram criados 100 casos de testes referentes à janela de cadastro de "Contatos".

Como não foram realizados os testes manuais dos 100 casos de testes, o tempo que se lavaria para executar os mesmo foi obtido através de estimativa, onde foi estimado um tempo aproximado de 1 hora.

Depois de cadastrados os 100 casos de testes na ferramenta, os mesmos foram colocados em execução, onde se obteve um tempo aproximado de 3 minutos.

Além dos 3 minutos para ter um embasamento mais preciso do tempo gasto com a automação dos 100 casos de testes, também foram levados em consideração os tempos gastos com planejamento, criação dos scripts, cadastros dos casos de testes, entre outros. Abaixo na tabela 1, é detalhado o tempo que foi utilizado em cada item.

ITEM	TEMPO (MINUTOS)
Planejamento	30
Criação da janela	45
Criação do script	20
Criação dos 100 casos de testes	120
Execução dos 100 casos de testes	3
Total:	218 (3,6 horas)

Tabela 1 – Detalhamento dos tempos gastos com a automação de testes.

Assim chegou-se a conclusão que inicialmente até se desenvolver o *script*, a janela e criar os casos de testes, pode ser que o tempo de desenvolvimento seja maior do que se processo fosse testado manualmente. Mesmo assim, conforme já foi dito no decorrer desse projeto, o grande ganho da automação de testes está nos testes regressivos, onde após um caso de teste ser desenvolvido, este pode ser reexecutado N vezes em diversas versões do software a ser testado, onde a partir de uma determinada execução começará a ser mais lucrativa a automação de testes do que os testes manuais. Vale lembrar também que os testes manuais são muito mais sujeitos a falhas do que os automatizados.

7. Conclusão

Embora a automação de testes seja vista como um elemento importante para melhorar a produtividade no ambiente de testes, as atividades mais propicias para o uso de tal técnica são aquelas que envolvem a execução de tarefas repetitivas e exaustivas facilmente sucessíveis a erros humanos, ou difíceis de serem realizadas manualmente. Assim a automação de testes em hipótese alguma deve substituir os testes manuais, devendo esta ser introduzida como uma técnica adicional, cujo objetivo principal é aumentar a qualidade do produto a ser entregue.

Conforme já foi dito no decorrer de todo projeto, a automação de testes envolve um grande tempo de análise, planejamento, desenvolvimento e execução. Sendo que inicialmente se perde mais tempo criando do que se realizando os testes. Onde o grande potencial da automação se encontra nos testes regressivos, pois uma vez criado um caso de teste, esse ficará disponível para ser executado enumeras vezes. Ou seja, a automação de testes é um investimento que cujo retorno será em longo prazo.

Dessa forma esse projeto procurou criar um software que auxilia-se o processo de desenvolvimento e manutenção dos casos de testes automatizados, procurando diminuir o tempo de criação, planejamento e manutenção, tendo como base as técnicas *Data-Driven* e *Keyword-Driven*.

Como trabalhos futuros, pretende-se expandir o software desenvolvido também para outros sistemas operacionais, e também possibilitar a integração do software com componentes visuais mais complexos, possibilitando assim uma maior integração com os softwares disponíveis atualmente no mercado.

Referências

FEWSTER, Mark; GRAHAM, Dorothy. Software Test Automation: effective use of test execution tools. Inglaterra, Londres: Addison-Wesley, 1999.

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

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