Testware: ferramenta de planejamento e execução de casos de teste

Fabiane Barreto Vavassori Benitti[1, 2], Ana Paula Zimmermann[1]

[1]Centro de Ciências Tecnológico da Terra e do Mar
Universidade do Vale do Itajaí, (UNIVALI) – Itajaí, SC – Brazil

[2]Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brazil

fabiane.benitti@univali.br, anapzi@gmail.com

Resumo. *Teste de software é um dos processos do ciclo de desenvolvimento do software que possui o objetivo de exercitar um programa a fim de descobrir erros, contribuindo para a qualidade do produto final. Por ser um processo complexo consome bastante recursos. Sendo assim, as ferramentas para automação de teste de software, permitem reduzir o tempo gasto e conseqüentemente os custos para a organização que as incorpora. Testware é uma ferramenta para automação de testes de caixa preta, que visa auxiliar na definição de casos de teste e sua execução.*

1. Introdução

Teste é uma das áreas da Engenharia de Software e uma das etapas do ciclo de desenvolvimento de software, sendo definida por Pressman (2005) como o processo de executar um programa com a intenção de descobrir um erro. Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto. O teste auxilia na produtividade e fornece evidências da confiabilidade e da qualidade do software. Uma organização gasta 40% do esforço do projeto total somente na etapa de teste, e se os defeitos forem descobertos na fase de manutenção, o custo por correção pode ser de 60 a 100 vezes maior [Pressman 2005].

Assim, surgem no mercado ferramentas para automatizar as diversas atividades inerentes a um processo de testes de software. Segundo Mats (2001) um dos principais problemas nas atividades de testes de software é a ausência de critérios para seleção dos casos de teste, definição da sua completude e estabelecimento de um ponto de parada, dificultando a revelação de falhas no produto. A ferramenta Testware, apresentada neste artigo, tem seu foco de atuação no problema relatado por Mats, auxiliando na definição e automação de casos de teste baseada nos conceitos da técnica de caixa preta que, segundo Bartié (2002), concentram-se nos requisitos funcionais do sistema.

Sendo assim, este artigo está organizado da seguinte forma: na seção 2 são apresentados os conceitos que embasam a ferramenta. Na seção 3 consta a especificação da ferramenta e descrição de sua funcionalidade. A seção 4 aborda ferramentas de automação de teste visando uma análise comparativa. Por fim, a seção 5 expõe as considerações finais e melhorias previstas para a ferramenta.

2. Teste de Software: caixa preta

Segundo Bartié (2002), os testes "têm por objetivo identificar o maior número possível de erros tanto nos componentes isolados quanto na solução tecnológica como um todo".

Desta forma, tem-se duas abordagens para os testes: testes de caixa branca e os testes de caixa preta.

Os testes de caixa branca são considerados “testes em pequeno porte” (*testing in the small*), ou seja, são tipicamente aplicados a componentes de programa pequenos (por exemplo, uma classe ou um método). Os testes de caixa preta, por outro lado, ampliam o foco e poderiam ser denominados “testes em grande porte” (*testing in the large*) [Pressman 2005]. Utilizando técnicas de caixa branca pode-se derivar casos de teste que: (1) garantam que todos os caminhos do código tenham sido exercitados ao menos uma vez; (2) exercitam todas as condições lógicas, verdadeiro e falso; (3) exercitam os ciclos nos seus limites e dentro de seus intervalos operacionais; e (4) exercitam as estruturas de dados internas para garantir sua validade. Os métodos mais conhecidos para a abordagem de caixa branca são: Teste de Caminho Básico e Teste de Estrutura de Controle.

Os testes de caixa preta são projetados para validar os requisitos funcionais, sem se preocupar com o funcionamento interno de um programa. As técnicas de teste de caixa preta concentram-se no domínio de informações do software, derivando os casos de teste ao dividir a entrada e a saída de uma maneira que proporcione uma satisfatória cobertura de teste. Existem quatro principais métodos utilizados para detectar casos de teste de caixa preta, que procuram aumentar a cobertura dos testes: particionamento por equivalência, análise de valor limite, técnicas de grafo de causa efeito e testes de comparação. A ferramenta Testware aborda as duas primeiras técnicas.

“A partição de equivalência é um método que divide o domínio de entrada de dados em classes (grupos de valores). Cada classe representa um possível erro a ser identificado, permitindo que os casos de testes redundantes de cada classe identificada sejam eliminados sem que a cobertura dos cenários existentes seja prejudicada.” [Bartié 2002].

As classes devem ser definidas de acordo com as seguintes diretrizes:

1. Se uma condição de entrada especifica um intervalo, uma classe de equivalência válida e duas inválidas são definidas;

2. Se uma condição de entrada exige um valor específico, uma classe de equivalência válida e duas inválidas são definidas;

3. Se uma condição de entrada especifica um membro de um conjunto, uma classe de equivalência válida e uma inválida são definidas; e

4. Se uma condição de entrada é booleana, uma classe de equivalência válida e uma inválida são definidas.

A análise de valor limite complementa o método de particionamento por equivalência, porém foca os casos de testes nas fronteiras de cada classe, por exemplo, o software de digitação das médias finais dos alunos de graduação, o campo nota permite a entrada de valores na seguinte faixa “x >= 0 and x <=10”. Desta forma, este método sugere os casos de teste com o valor –1, 11, 0 e 10.

Este princípio foi criado, pois um grande número de erros é encontrado nas fronteiras do domínio de entrada. Além de focalizar somente as condições de entrada, a

análise de valor limite também deriva casos de teste para o domínio de saída, apresentando as seguintes diretrizes para a criação de casos de teste:

1. Se uma condição de entrada especifica um intervalo limitado pelos valores a e b, devem-se criar casos de teste com valores imediatamente acima e abaixo de a e b;

2. Se uma condição de entrada especifica vários valores, casos de teste deverão ser realizados para exercitar os números mínimos e máximos permitidos, também valores imediatamente acima e abaixo do mínimo e máximo selecionado;

3. As diretrizes 1 e 2 também são aplicadas às condições de saída. Por exemplo, uma tabela de temperatura *versus* pressão é esperada como saída de um programa de análise de engenharia. Devem ser projetados casos de testes para criar um relatório de saída que produza o número máximo e mínimo aceitável de entradas na tabela; e

4. Se o programa contém uma estrutura interna que existem limites prescritos, deve realizar testes a fim de exercitar esta estrutura de dados no seu limite, por exemplo, um vetor de 100 entradas.

3. Especificação e Apresentação da Ferramenta Testware

Para a execução de testes de software com a ferramenta Testware é necessário seguir as etapas ilustradas na Figura 1. Na etapa 1 é preciso configurar a ferramenta descrevendo o roteiro de execução da aplicação em teste. Na etapa 2, de acordo com as diretrizes dos métodos de caixa preta (particionamento por equivalência e análise de valor limite), o testador fornece os valores que a ferramenta irá utilizar para testar a aplicação. Após, na etapa 3, através do roteiro de execução e dos valores definidos anteriormente a ferramenta gera os casos de teste. Na etapa 4 a ferramenta executa automaticamente a aplicação em teste. E, por fim, o testador informa o resultado do teste executado, ficando os casos de teste disponíveis na ferramenta para reprodução.

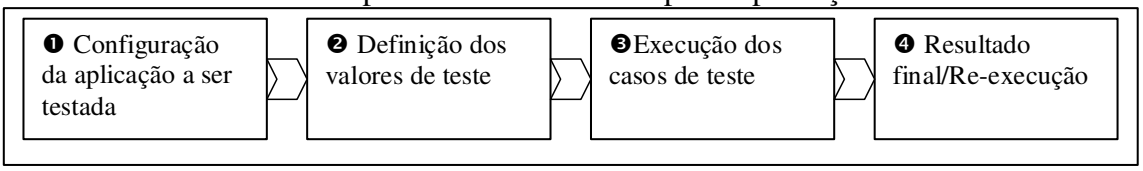

Figura 1. Etapas da ferramenta

Considerando o processo descrito, as principais funcionalidades da ferramenta desenvolvida são demonstradas através do diagrama de casos de uso ilustrado na Figura 2.

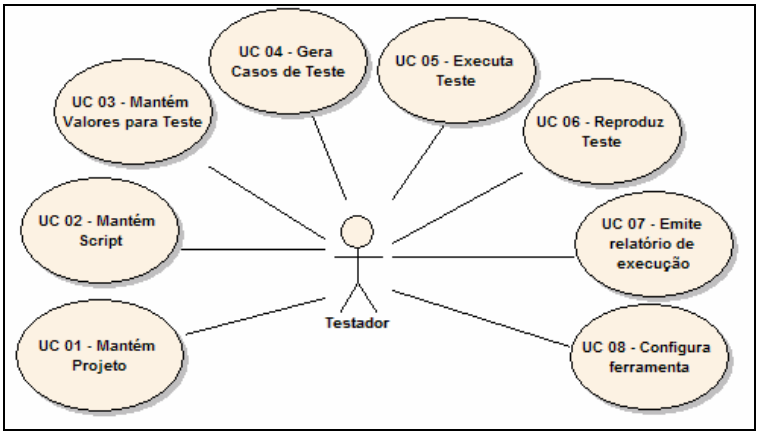

Figura 2. Modelo de Casos de Uso

Sucintamente, os casos de uso possuem os seguintes comportamentos:

- ✓ O caso de uso 01 refere-se ao cadastramento do projeto testado, mantendo informações como o nome do projeto e indicação do caminho da aplicação a testar (*path* do executável).
- ✓ No caso de uso 02 é onde ocorre a edição e a validação do *script*. O *script* é o roteiro de execução da aplicação em teste, e possui uma sintaxe própria (conforme detalhado na seção 3.1). A Figura 3a, apresenta um exemplo de *script*.
- ✓ O caso de uso 03 prevê que o testador poderá informar valores de teste para cada campo de entrada definido no *script*. Conforme o tipo do campo: valor, intervalo, conjunto ou booleano - são exibidas dicas, instruindo o testador a cadastrar valores de teste válidos e inválidos, conforme as diretrizes dos métodos particionamento por equivalência e análise de valor limite (conforme pode ser observado na Figura 3b).

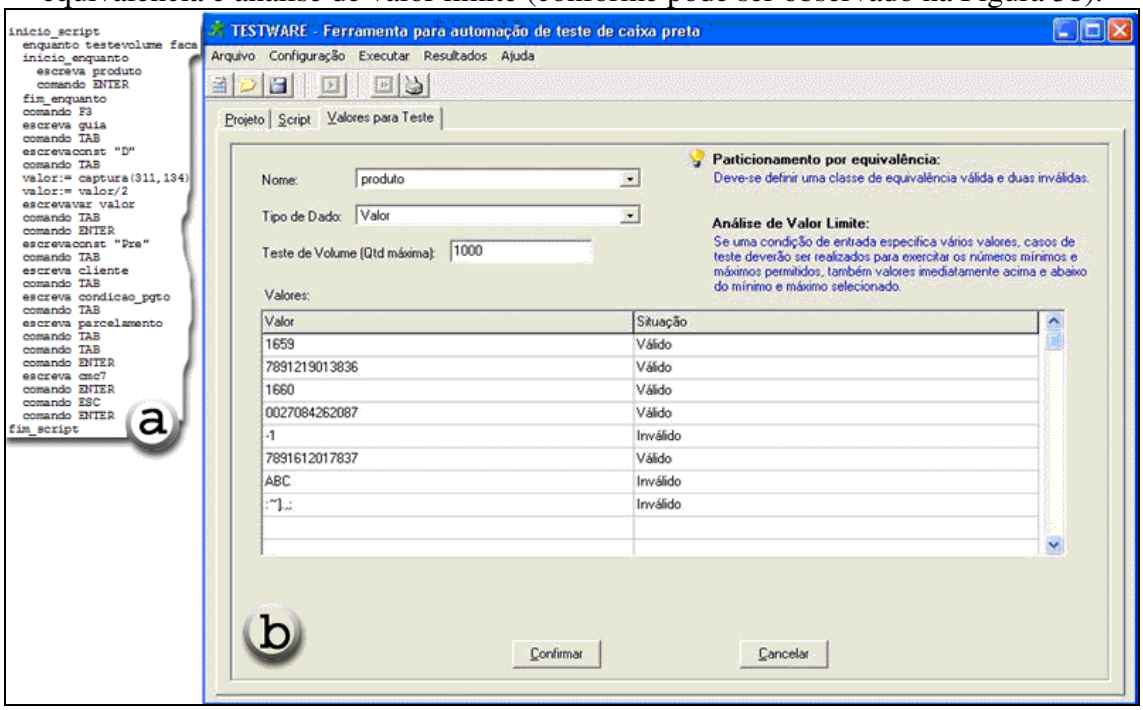

Figura 3. Interface para cadastro de valores de teste por campo

Além disso, os aplicativos testados que possuam campos iterativos (por exemplo, funcionalidade de vendas, na qual se pode relacionar vários produtos em uma venda) poderá ser aplicado o teste de volume. Conforme Bartié (2002), o teste de volume determina o limite de processamento de um aplicativo através do incremento do volume de operações. Para isto, os campos iterativos devem ser identificados no *script* e também informados o valor máximo de iterações a ser aplicado (campo "Teste de volume" na interface da Figura 3b). Desta forma, será criado um caso de teste separadamente para aplicar o teste de volume.

- ✓ O caso de uso 04, através do *script* e dos valores cadastrados para teste, realiza a geração automática dos casos de teste. Os casos são gerados pela ferramenta observando a seguinte regra: no primeiro caso de teste em cada campo de entrada será inserido um valor válido, após serão criados casos de testes visando exercitar as opções inválidas de cada campo, sendo para os demais campos utilizados sempre valores válidos. A ferramenta irá gerar casos de teste para todos os valores

cadastrados, sempre exercitando um campo de cada vez e para os demais aplicando valores válidos.

- ✓ O caso de uso 05 descreve a possibilidade de execução dos casos de testes permitindo ao testador acompanhar a execução e, ao finalizar a execução, informar o resultado do teste. A Figura 4 ilustra a interface para este procedimento.

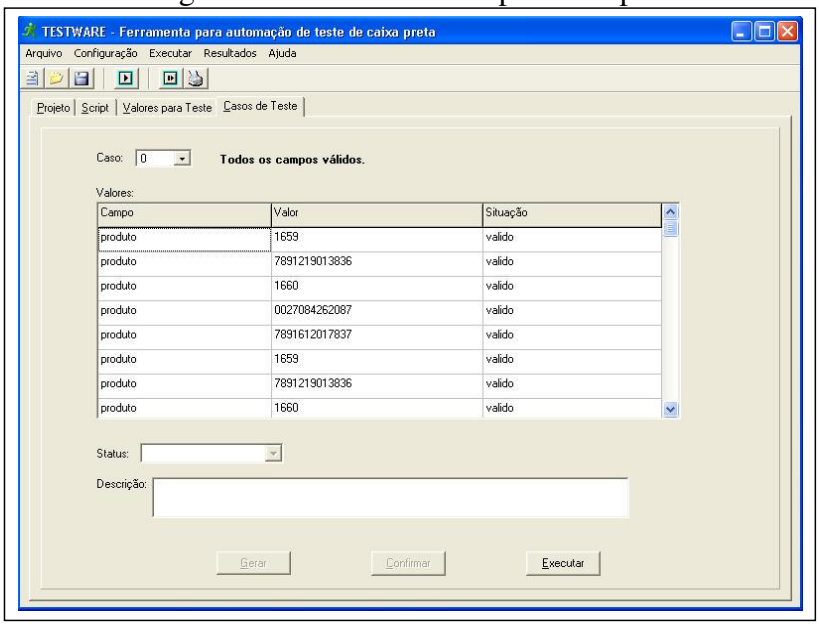

Figura 4. Interface para execução de um caso de teste

- ✓ O caso de uso 06 apresentará o resultado dos testes realizados, permitindo ao testador, a qualquer momento, reproduzir um caso de teste executado anteriormente (visando abranger os testes de regressão[1]). A Figura 5 retrata a interface de acompanhamento e os resultados possíveis de um caso de teste:
    - Sucesso: o caso de teste foi concluído com sucesso, considerando valores válidos para todos os campos de entrada; ou o caso de teste não foi concluído, pois se utilizou valores inválidos em algum campo.
    - Erro: o caso de teste foi concluído, tendo utilizado valor inválido; ou o caso de teste não concluiu, utilizando todos os valores válidos.
    - *Timeout*: a aplicação travou.

[1] "O teste de regressão é a reexecução de algum subconjunto de testes que já foram conduzidos para garantir que as modificações não propagaram efeitos colaterais indesejáveis" [Pressman 2005].

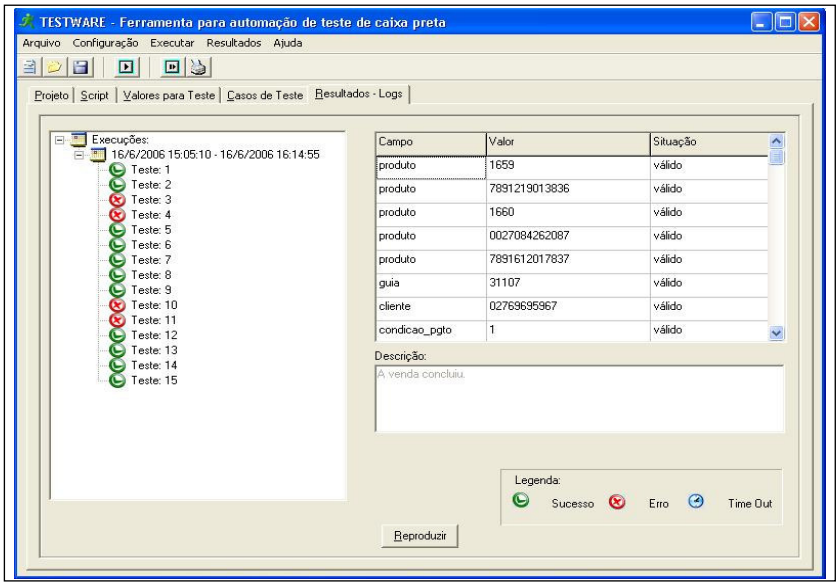

Figura 5. Interface de acompanhamento dos testes executados

- ✓ Através do caso de uso 07 é emitido um relatório de execução informando os casos de teste e o seu resultado. Este relatório deverá ser repassado aos programadores para efetuar correções no sistema.
- ✓ O caso de uso 8 permite ao testador configurar: (i) o conjunto de teclas utilizadas e seus devidos códigos ASCII (*American Standard Code for Information Interchange*); (ii) o tempo de espera para a aplicação em teste carregar e; (iii) o intervalo entre o envio de dados de um campo para outro.

A Figura 6 retrata o diagrama de classes de domínio da ferramenta Testware.

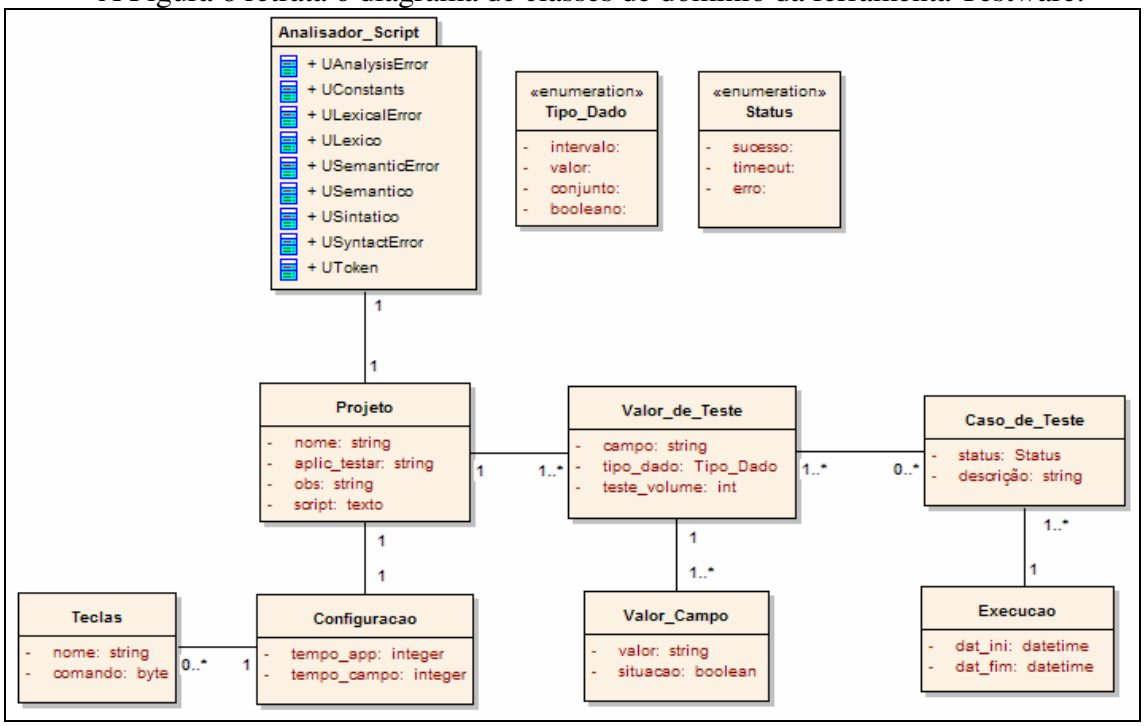

Figura 6. Diagrama de Classes

O diagrama de classes da ferramenta possui sete classes principais relacionadas entre si, conforme detalhamento:

- ✓ Projeto: é a classe principal da ferramenta, contendo informações gerais referente ao projeto;
- ✓ Valor de Teste: esta classe possui informação referente aos campos de entrada da aplicação em teste;
- ✓ Valor_Campo: valores a serem testados para cada campo de entrada da aplicação;
- ✓ Execução: é a classe que possui dados sobre a execução do teste;
- ✓ Caso_de_Teste: esta classe contém o status e considerações referente a execução do teste, bem como a referência aos valores testados no caso;
- ✓ Configuração: possui as configurações gerais da ferramenta, como: tempo para a aplicação em teste carregar e intervalo de tempo para envio de dados de um campo para outro;
- ✓ Teclas: classe com o código ASCII correspondente aos comandos utilizados no script, por exemplo: enter, F1, F2, tab, etc.

3.1 Estrutura do *Script*

O *script* tem o objetivo de apresentar o roteiro de execução da ferramenta a se testar. Sua gramática foi escrita e validada na ferramenta Gals [Gesser 2003], que gerou as classes ULexico e USintatico (pacote Analisador_Script da Figura 6) com a implementação do analisador léxico e sintático, respectivamente, que por sua vez foram incorporados a ferramenta. A Tabela 1 exibe as principais instruções para construção do *script*.

Tabela 1. Instruções da gramática do *script*

Instrução	Descrição
inicio_script	Primeira instrução do *script* e indica que ele foi inicializado, esta instrução é particularmente importante para permitir a implementação de diversos *scripts* em um mesmo projeto.
enquanto testevolume faca	Esta instrução deve ser utilizada para campos iterativos, por exemplo, o campo código de produto, em um sistema de caixa, na validação do *script* serão identificados os campos de entrada que estiverem dentro deste laço e solicitado o cadastrado da quantidade máxima de iterações desejadas para a aplicação do teste de volume, e para os demais casos de teste será realizado um *random* de dez.
inicio_enquanto	Identifica que o laço "enquanto" foi iniciado.
escreva	Declara os campos de entrada existentes no aplicativo em teste.
comando	Define a tecla utilizada para navegação entre os campos, por exemplo: "tab", "enter". Os códigos ASCII destas teclas devem estar devidamente cadastrados na ferramenta.
fim_enquanto	Marca o final do laço enquanto.
Escrevaconst	Instrução utilizada para enviar à aplicação valores/texto pré-definidos.
captura(x,y)	Função que retorna uma string contendo o valor / texto de determinada posição da tela.
:=	Atribui um valor a uma determinada variável.
escrevavar	Envia a aplicação o valor de uma variável interna do *script*.
ALT#A	Envia uma combinação de teclas para a aplicação.
fim_script	Identifica o término do *script*.

4. Estado da Arte

O aumento das exigências dos usuários e concorrência cada vez maior na área de desenvolvimento de software vem pressionando as empresas a aperfeiçoarem seus processos. No entanto, considerando a complexidade envolvida nos sistemas atuais, para se testar adequadamente um software, uma organização gasta 40% do esforço de projeto total em teste (Pressman, 2005). Para diminuir o tempo e custo do processo de teste de software algumas ferramentas propõem automatizar as atividades relacionadas com este processo, sendo algumas destas ferramentas descritas nesta seção.

4.1 TestComplete 3.11

A TestComplete é uma ferramenta de automação de testes, aplica testes de unidade, funcional e de regressão. É desenvolvida pela empresa AutomatedQA Corporation, fundada em 1999 com matriz nos Estados Unidos [AutomatedQA, 2005].

As principais funcionalidades da ferramenta são:

- ✓ Grava a interação do usuário com a interface do software, através dos eventos do mouse e teclado, gerando *scripts* em Delphi, Visual Basic e C;
- ✓ Permite criar ou alterar um *script* de teste;
- ✓ Reconhece variáveis e componentes da interface;
- ✓ Permite realizar teste de carga e de escalabilidade de aplicações web;
- ✓ Permite gerar o resultado dos testes em arquivo com extensão HTML (*Hyper Text Markup Language*) e XML (*Extensible Markup Language*), podendo assim incluir na documentação do software;
- ✓ Permite depurar o *script* incluindo *breakpoints*; e
- ✓ Realiza testes em interfaces *desktop* e *web*.

4.2 QuickTest Professional 8.2

Esta ferramenta pertence a empresa Mercury Intercative Corporation, fundada em 1989 [Mercury 2005]. A QuickTest é um módulo do suíte de testes que contém mais dois módulos:

- ✓ TestDirector: é o módulo gerenciador de testes e defeitos, controla o processo de testes de "ponta-a-ponta", garantindo que todos os requisitos dos usuários sejam cobertos pelos casos de testes construídos; e
- ✓ LoadRunner: responsável pela automação de testes de desempenho, atendem aos testes de carga e volume, a ferramenta permite a criação de um cenário para a execução dos testes e monitora *on-line* os elementos envolvidos, como: servidores, banco de dados e rede.

A QuickTest é uma ferramenta de automação de testes funcionais, ela grava a ação desejada na aplicação, parametriza e em seguida executa o teste. Suas principais características e funcionalidades são:

- ✓ Gera script da interação do usuário em VBScript;
- ✓ Permite incluir ou alterar o *script*;

- ✓ Realiza testes em interfaces *web* e *desktop*;
- ✓ Possibilita a depuração do *script*;
- ✓ Reconhece variáveis e componentes da interface;
- ✓ Após a execução apresenta um *log* com o resultado do teste;
- ✓ Permite cadastrar o executável ou *browser* a ser testado, desta forma o executa automaticamente;
- ✓ Permite cadastrar parâmetros de entrada e saída; e
- ✓ Criptografa as senhas digitadas nas interfaces.

4.3 J-Fut

Esta é uma ferramenta acadêmica, apresentada no 19º Congresso Brasileiro de Engenharia de Software. A J-Fut possui o objetivo de apoiar o teste funcional de programas desenvolvidos em Java, possuindo as seguintes características e funcionalidades [Rocha et al. 2005]:

- ✓ Oferece suporte as técnicas de particionamento de equivalência, análise de valor limite e este funcional sistemático ;
- ✓ Permite que pré e pós condições do sistema sejam avaliadas;
- ✓ Permite efetivar as atividades básicas para a aplicação de critérios: instrumentação, seleção, execução de casos de testes e análise de cobertura;
- ✓ Realiza teste funcional de unidade;
- ✓ Analisa o comportamento interno do programa identificando classes de equivalência de uma operação;
- ✓ Não há acesso direto ao programa em teste e sua execução sempre é feita a partir dos casos de teste implementados; e
- ✓ Os testes são feitos a partir de um projeto de teste, que armazena todos os dados relativos ao teste, incluindo seu nome, classes e métodos em teste, bibliotecas necessárias, requisitos de teste, condições de entrada e registro de execução.

4.4 Análise Comparativa

A Tabela 2 mostra uma análise comparativa entre as três ferramentas estudadas: TestComplete, QuickTest e J-Fut, e a ferramenta desenvolvida neste projeto, a TestWare.

Tabela 2. Análise comparativa entre ferramentas similares

Características	TestComplete	QuickTest	J-Fut	TestWare
Realiza testes de quais fases (Unidade, Integração, Sistema e Aceitação)	- Teste de unidade - Teste de integração	- Teste de unidade - Teste de integração	- Teste de unidade	- Teste de integração
Tipos de Teste (Caixa Branca e Caixa Preta)	Caixa Preta	Caixa Preta	- Caixa Preta e Branca	- Caixa Preta
Método de caixa preta abordado	- nenhum	- nenhum	- Particiona-mento de equivalência - Teste funcional sistemático - Análise de valor limite	- Particionamento de equivalência - Análise de valor limite
Categorias de Testes	- funcional - regressão - carga - escalabilidade	- funcional - volume - carga	- funcional	- funcional - volume - regressão
Classificação da ferramenta	- execução dos testes	- execução dos testes	-desenvolvimento dos testes - execução dos testes	- desenvolvimento dos testes - execução dos testes
Tipo de aplicações testadas	-web - *desktop*	-web - *desktop*	- *desktop* escritas em Java	- *desktop* indepentente da linguagem
Formato de entrada dos casos de testes	- *script*	- *script*	- base de casos	- base de casos - *script*
Resultado ao usuário	- interface e gera arquivo de *log* em HTML e XML	- interface e gera arquivo de *log*	- interface	- interface

A partir da comparação percebe-se que a TestComplete e a QuickTest apresentam praticamente as mesmas características. As duas gravam a iteração do usuário com a interface em *script*, após executa o teste, e apresenta relatórios de *logs*. Estas ferramentas são comerciais e apresentam um custo elevado. Possuem versão *trial* para *download*, para a realização desta análise comparativa foram aplicados diversos testes, com aplicações *web* e *desktop*.

Já a ferramenta acadêmica J-Fut implementa dois métodos de caixa preta, o método de particionamento de equivalência e o método de análise de valor limite, através destes métodos gera uma base de casos de testes e os executa. Uma grande desvantagem da J-Fut é testar somente aplicações implementadas na linguagem Java.

Com a pesquisa das ferramentas de automação de testes, percebe-se que existem algumas opções já em comercialização ou em desenvolvimento através de projetos de pesquisa, porém, a Testware apresenta um diferencial em relação aos produtos em comercialização: uma vez descrito o roteiro de interação (*script*) o testador informa os valores e os casos de testes são gerados automaticamente, ou seja, a partir do mesmo *script* tem-se diversos casos de testes (situação não permitida nas ferramentas analisadas). Outro aspecto a destacar refere-se ao relatório (já formatado para ser

encaminhado aos programadores), pois nas demais ferramentas comerciais o testador gera o relatório a partir da análise dos *logs* da ferramenta.

5. Considerações Finais

Percebe-se que a área de teste de software vem sendo explorada cada vez mais, tendo em vista os seguintes aspectos: os sistemas tornam-se cada vez mais complexos; a concorrência entre empresas da área de desenvolvimento de software é cada vez maior; e os clientes tornam-se mais exigentes.

Dias Neto e Travassos (2006) ressaltam que "apesar da importância das atividades de planejamento e controle de testes de software, observa-se na literatura técnica da área de testes pouca importância dada a elas. Poucas abordagens que apóiam a essas atividades são propostas, e normalmente para um contexto específico, e um número mais reduzido ainda é aplicado na indústria, caracterizando o estado da arte das atividades de planejamento e controle dos testes." Tendo em vista este relato, a ferramenta Testware apresenta uma abordagem para este problema, atuando na definição e execução de casos de testes.

A validação da ferramenta Testware[2] foi realizada através do módulo de caixa do sistema ItlSys, pertencente a empresa Intelidata Ltda, que constitui um sistema de informação ERP (*Enterprise Resource Planning*) de grande porte, destinado a área comercial. A Testware também foi validada através do aplicativo SAPRO (Sistema de Acompanhamento de Processos), desenvolvido pela empresa Four-M Comercial e Serviços de Informática Ltda, tendo como objetivo automatizar as rotinas de escritórios de advocacia. Alguns aspectos detectados para melhoria dos aplicativos testados foram:

- ✓ no cadastro de clientes do SAPRO o campo relativo a data de cadastro aceita, sem nenhuma mensagem de confirmação, datas maiores que a data atual;
- ✓ no registro de venda do ItlSys foram aplicados valores inválidos para o campo de entrada "guia" e em dois casos a venda finalizou;
- ✓ foram inseridos valores inválidos para o campo "parcelamento" do ItlSys, porém, cada um obteve um erro diferente. No primeiro, foi enviado a aplicação o valor "02566985", o caixa exibiu o seguinte erro "*Run-time error 6 Overflow*", significando que o valor inserido contém mais caracteres do que foi definido no banco, sendo que após o erro o aplicativo do caixa fechou deixando a operação da venda não concluída. No segundo caso, foi enviado ao campo "parcelamento" o valor "abc", e a aplicação aceitou normalmente, finalizando a venda como cheque a vista (enquanto o comportamento esperado seria um aviso de parcelamento inválido).

Através da validação foi possível relatar algumas inconformidades nos aplicativos testados, bem como identificar melhorias para aperfeiçoar a ferramenta Testware, como a inclusão de gravação de macros, controle de sub-projetos, automatização do resultado da execução, dentre outros.

Referências

AutomatedQA. (2005) TestComplete. Disponível em: <http://www.automatedqa.com/products/testcomplete/>. Acesso em: 08 set. 2005.

[2] Em http://www.inf.furb.br/~fabiane está disponível um vídeo demonstrativo das principais funcionalidades da ferramenta.

Bartié, A. (2002) “Garantia da qualidade de software” , Campus.

Dias Neto, A.C. e Travassos, G.H. (2006) “Maraká: infra-estrutura computacional para apoiar o planejamento e controle dos testes de software.” In *Simpósio Brasileiro de Qualidade de Software*, p. 250 – 264.

Gesser, C. E. (2003) “GALS: Gerador de analisadores léxicos e sintáticos.” Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)–Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis.

Mats, L. (2001) “The top five software-testing problems and how to avoid them”, EDN Europe, Fevereiro, Vol. 46 Issue 2, p 37.

Mercury. (2005) **QuickTest**. Disponível em: <http://www.mercury.com/us/products/quality-center/functional-testing/quicktest-professional/>. Acesso em: 10 set. 2005.

Pressman, R. S. (2005) “Software Engineering: A Practitioner’s Approach”, McGraw-Hill, 6th ed, New York.

Rocha, A. D.; Simão, A. S.; Maldonado, J. C.; Masiero, P. C. (2005) “Uma ferramenta baseada em aspectos para o teste funcional de programas Java”. In: *Congresso Brasileiro de Engenharia de Software*. Minas Gerais. Disponível em: < http://www.sbbd-sbes2005.ufu.br/arquivos/17-%209610.pdf>. Acesso em: 5 set. 2005.