AutoTest – Um *Framework* Reutilizável para a Automação de Teste Funcional de Software

Marcelo Fantinato[1, 2], Adriano C. R. da Cunha[1, 2], Sindo V. Dias[1, 2], Sueli A.Mizuno[1], Cleida A. Q. Cunha[1]
1. CPqD Telecom & IT Solutions / DSB – Diretoria de Soluções em *Billing*
2. UNICAMP – Universidade Estadual de Campinas / IC – Instituto de Computação
Campinas – SP – Brasil
{marcelof, adrcunha, sindo, sueliam, cleida}@cpqd.com.br

Resumo

O teste de software é uma atividade de alto impacto no processo de desenvolvimento de sistemas de grande porte. A automação de parte do teste tem sido vista como a principal medida para melhorar a eficiência desta atividade. Entretanto, o sucesso da aplicação de uma abordagem automatizada depende da utilização de uma estratégia sistemática. Este artigo apresenta um *framework* reutilizável para a automação de teste funcional de software, chamado AutoTest, cuja aplicação visa a obtenção de reais ganhos com a automação. Além disso, são apresentados os resultados da aplicação do *framework* proposto na automação de teste de um sistema desenvolvido por uma empresa de telecomunicações.
Palavras-chave: teste de software; teste funcional; automação de teste.

Abstract

Software testing is an activity with great effect on the development process of large systems. Automation has been seen as the main way to improve the testing efficiency. However, the success of an automatized approach depends on using a systematic strategy. This paper presents a reusable framework for the software functional testing automation, called AutoTest, whose application aims at the achievement of real benefits with the automation. Moreover, the application results of the proposed framework on testing automation of a system developed by a telecommunication company are presented.
Keywords: software testing; functional testing; testing automation.

1. Introdução

O teste de software é uma das principais atividades realizadas para melhorar a qualidade de um produto em desenvolvimento. Seu principal objetivo é revelar a presença de erros no software o mais cedo possível no ciclo de desenvolvimento de software, buscando minimizar o custo da correção dos mesmos. Esta atividade tem apresentado progressivamente um maior grau de abrangência e de complexidade dentro do processo de desenvolvimento [12, 16, 17].

Embora o teste de software seja uma atividade bastante complexa, geralmente ela não é realizada de forma sistemática devido a uma série de fatores como limitações de tempo, recursos e qualificação técnica dos envolvidos. Outros agravantes para a realização dessa atividade são a alta complexidade dos sistemas sendo atualmente desenvolvidos e a constante necessidade de sua rápida evolução.

A automação de parte do teste de software tem sido vista como a principal medida para melhorar a eficiência dessa atividade, e várias soluções têm sido propostas para esta finalidade. A automação do teste consiste em repassar para o computador tarefas de teste de software que seriam realizadas manualmente, sendo feita geralmente por meio do uso de ferramentas de automação de teste. Podem ser consideradas para a automação as atividades de geração e de execução de casos de teste [2, 8].

Quando executada corretamente, a automação de teste é uma das melhores formas de reduzir o tempo de teste no ciclo de vida do software, diminuindo o custo e aumentando a produtividade do desenvolvimento de software como um todo, além de, conseqüentemente,

aumentar a qualidade do produto final. Estes resultados podem ser obtidos principalmente na execução do teste de regressão, que se caracteriza pelo teste de aplicativos já estáveis que passam por uma correção de erros, ou de aplicativos já existentes que são evoluídos para uma nova versão e suas funcionalidades são alteradas [8].

Apesar de haver um consenso, entre os especialistas, dos ganhos que podem ser alcançados com o uso de uma boa estratégia de automação de teste, esta é uma área ainda pouco dominada pela indústria de software. Deste modo, as empresas acabam atuando na automação de teste sem a definição de objetivos e expectativas claros e reais e sem a aplicação de técnicas apropriadas. Por conseqüência, têm-se constatado um grande número de insucessos nos esforços para a automação de teste [1, 3, 5, 6, 11].

Apesar de existirem ferramentas comerciais que oferecem suporte à automação de teste de software, estas são geralmente limitadas em relação às funcionalidades oferecidas [7, 22, 23, 24, 25]. A aplicação pura destas ferramentas tem demonstrado uma fraqueza na aplicação sistemática de estratégias de teste visando um maior reaproveitamento de esforço [5, 6, 11]. Assim, existe a necessidade de uma abordagem de automação de teste que ofereça maiores funcionalidades e vantagens que o uso puro de ferramentas de automação de teste.

O propósito deste artigo é apresentar o *framework* AutoTest, um *framework* reutilizável para a automação da execução de teste funcional, amplamente aplicável em diferentes projetos de desenvolvimento de software. Este *framework* foi desenvolvido com base nas técnicas de automação *data-driven* e *keyword-driven*. Composto por um conjunto de ferramentas de software, scripts de teste, *templates* de planilhas, e regras e procedimentos de utilização, ele permite automatizar a execução de casos de teste com pouca necessidade de codificação, dependendo do projeto em questão. Com o seu uso é possível obter um conjunto de casos de teste automatizados que pode ser facilmente executado, re-executado e atualizado.

O escopo do teste tratado pelo *framework* AutoTest e, conseqüentemente neste artigo, é a execução de teste funcional (também chamado de teste de caixa-preta), no nível de teste de sistema. Assim, a execução de teste estrutural (também chamado de teste de caixa-branca), nos níveis de teste de unidade e de integração, não é tratada aqui. Além disso, a geração de casos de teste, tanto para teste funcional como para teste estrutural, também não é tratada.

Este artigo está organizado da seguinte forma: na Seção 2 são descritas as principais técnicas aplicadas na automação de teste, incluindo as técnicas utilizadas no desenvolvimento do *framework* apresentado; na Seção 3 é apresentado o *framework* reutilizável AutoTest para a automação de teste de software; na Seção 4 é apresentado um estudo de caso em que o *framework* definido é aplicado em um sistema real; e, finalmente, na Seção 5 são apresentadas a conclusão e sugestões de trabalhos futuros.

2. Técnicas de Automação de Teste

As principais técnicas de automação de teste apresentadas na literatura são: *record & playback*, programação de scripts, *data-driven* e *keyword-driven*. Esta seção apresenta uma breve descrição de cada uma destas técnicas com o objetivo de oferecer um melhor entendimento do *framework* AutoTest descrito na próxima seção.

A técnica *record & playback* consiste em, utilizando uma ferramenta de automação de teste, gravar as ações executadas por um usuário sobre a interface gráfica de uma aplicação e converter estas ações em scripts de teste que podem ser executados quantas vezes for desejado. Cada vez que o script é executado, as ações gravadas são repetidas, exatamente como na execução original. Para cada caso de teste é gravado um script de teste completo que inclui os dados de teste (dados de entrada e resultados esperados), o procedimento de teste (passo a passo que representa a lógica de execução) e as ações de teste sobre a aplicação.

A vantagem da técnica *record & playback* é que ela é bastante simples e prática, sendo uma boa abordagem para testes executados poucas vezes. Entretanto, são várias as desvantagens desta técnica ao se tratar de um grande conjunto de casos de teste automatizados, tais como: alto custo e dificuldade de manutenção, baixa taxa de reutilização, curto tempo de vida e alta sensibilidade a mudanças no software a ser testado e no ambiente de teste. Como exemplo de um problema desta técnica, uma alteração na interface gráfica da aplicação poderia exigir a regravação de todos os scripts de teste [4, 5, 6, 11, 15, 19].

A técnica de programação de scripts é uma extensão da técnica *record & playback.* Através da programação os scripts de teste gravados são alterados para que desempenhem um comportamento diferente do script original durante sua execução. Para que esta técnica seja utilizada, é necessário que a ferramenta de gravação de scripts de teste possibilite a edição dos mesmos. Desta forma, os scripts de teste alterados podem contemplar uma maior quantidade de verificações de resultados esperados, as quais não seriam realizadas normalmente pelo testador humano e, por isso, não seriam gravadas. Além disso, a automação de um caso de teste similar a um já gravado anteriormente pode ser feita através da cópia de um script de teste e sua alteração em pontos isolados, sem a necessidade de uma nova gravação.

A programação de scripts de teste é uma técnica de automação que permite, em comparação com a técnica *record & playback*, maior taxa de reutilização, maior tempo de vida, melhor manutenção e maior robustez dos scripts de teste. No exemplo de uma alteração na interface gráfica da aplicação, seria necessária somente a alteração de algumas partes pontuais dos scripts de teste já criados. Apesar destas vantagens, sua aplicação pura também produz uma grande quantidade de scripts de teste, visto que para cada caso de teste deve ser programado um script de teste, o qual também inclui os dados de teste e o procedimento de teste. As técnicas *data-driven* e *keyword-driven*, que são versões mais avançadas da técnica de programação de scripts, permitem a diminuição da quantidade de scripts de teste, melhorando a definição e a manutenção de casos de teste automatizados [4, 6, 7, 18].

A técnica *data-driven* (técnica orientada 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.

A principal vantagem da técnica *data-driven* é que se pode facilmente adicionar, modificar ou remover dados de teste, ou até mesmo casos de teste inteiros, com pequena manutenção dos scripts de teste. Esta técnica de automação permite que o projetista de teste e o implementador de teste trabalhem em diferentes níveis de abstração, dado que o projetista de teste precisa apenas elaborar os arquivos com os dados de teste, sem se preocupar com questões técnicas da automação de teste [4, 6, 7, 13, 19].

A técnica *keyword-driven* (técnica orientada a palavras-chave) consiste em extrair, dos scripts de teste, o procedimento de teste que representa a lógica de execução. Os scripts de teste passam a conter apenas as ações específicas de teste sobre a aplicação, as quais são identificadas por palavras-chave. Estas ações de teste são como funções de um programa, podendo inclusive receber parâmetros, que são ativadas pelas palavras-chave a partir da execução de diferentes casos de teste. O procedimento de teste é armazenado em um arquivo separado, na forma de um conjunto ordenado de palavras-chave e respectivos parâmetros.

Assim, pela técnica *keyword-driven*, os scripts de teste não mantêm os procedimentos de teste no próprio código, obtendo-os diretamente dos arquivos de procedimento de teste. A principal vantagem da técnica *keyword-driven* é que se pode facilmente adicionar, modificar

ou remover passos de execução no procedimento de teste com necessidade mínima de manutenção dos scripts de teste, permitindo também que o projetista de teste e o implementador de teste trabalhem em diferentes níveis de abstração [4, 7, 9, 13, 19].

3. *Framework* AutoTest

O *framework* AutoTest possui como característica principal a sua reutilização na automação de teste funcional em diferentes projetos de desenvolvimento de software, visando a melhoria de suas produtividade e qualidade. Ele é formado por um conjunto de ferramentas de software, scripts de teste, *templates* de planilhas, e regras e procedimentos de utilização que não mudam em função de novos projetos. A elaboração do *framework* AutoTest foi feita com base, principalmente, em uma técnica mista de automação de teste, chamada de técnica *keyword-data-driven*, definida a partir de duas outras técnicas de automação conhecidas.

Uma das vantagens da aplicação deste *framework* é que, para novos projetos de teste de software, a infra-estrutura para sua automação já está montada. Assim, as atividades específicas ainda necessárias para o novo projeto de teste em questão são: a elaboração das planilhas de teste com os dados e procedimentos de teste; e a manutenção do *framework*, incluindo a elaboração de novos scripts de teste que sejam necessários.

3.1. Técnica *Keyword-Data-Driven*

Com base nas técnicas de automação de teste *data-driven* e *keyword-driven*, foi definida uma técnica mista de automação de teste que reúne as suas principais vantagens. Chamada de *keyword-data-driven* (orientada a dados e a palavras-chave), esta técnica possibilita fortemente a diminuição da quantidade de scripts de teste, melhorando ainda mais a definição e a manutenção de casos de teste automatizados.

A técnica *keyword-data-driven* consiste em extrair, dos scripts de teste, tanto os dados de teste quanto o procedimento de teste, os quais são armazenados em arquivos separados dos scripts de teste. Assim, os scripts de teste não mantêm nem os dados de teste nem o procedimento de teste no próprio código, obtendo-os diretamente destes arquivos, mantidos em planilhas eletrônicas. Para a elaboração destes arquivos, foram adotadas regras de sintaxe e de semântica claras que permitem seu fácil entendimento.

Na técnica *keyword-data-driven* existem dois tipos de planilhas: as planilhas de dados de teste (chamadas de planilhas *data-driven*) e as planilhas de procedimento de teste (chamadas de planilhas *keyword-driven*). A Figura 1 apresenta uma comparação das técnicas de automação, ilustrando as diferenças entre elas, em relação à quantidade de scripts de teste e ao tipo de informação que é extraída dos scripts de teste e mantida em planilhas separadas.

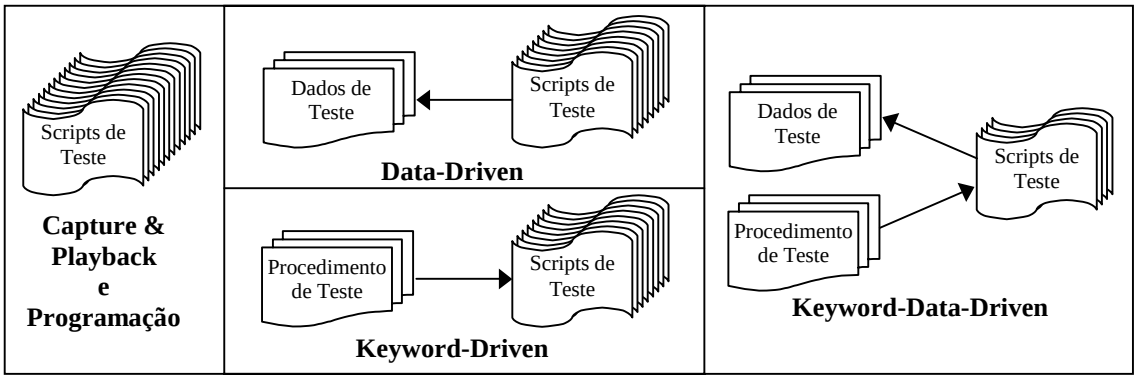

Figura 1 – Comparação entre as Técnicas de Automação de Teste

Os scripts de teste também são divididos em dois tipos: os scripts *data-driven*, que executam ações sobre a aplicação com base nos dados existentes nas planilhas *data-driven*; e os scripts *keyword-driven*, que executam ações mais genéricas, porém pontuais, sobre a aplicação. Os scripts *data-driven* são específicos e precisam ser criados ou alterados em função de novas planilhas *data-driven* existentes, enquanto que os scripts *keyword-driven* são genéricos e precisam ser programados apenas uma vez.

As planilhas *data-driven* especificam todos os dados de teste dos casos de teste, ou seja, todos os dados de entrada bem como os resultados esperados. A Figura 2 apresenta um exemplo de uma planilha *data-driven*. Cada linha da planilha, ou conjunto de linhas, descreve detalhadamente um caso de teste por meio de três grupos de colunas: o grupo "Caso de Teste" descreve brevemente um caso de teste, o qual pode ser dividido hierarquicamente em vários casos de teste derivados; o grupo "Dados de Entrada" descreve detalhadamente todos os dados a serem usados com entrada em sua execução; e o grupo "Resultados Esperados" descreve detalhadamente todos os dados a serem obtidos durante a execução. Este terceiro grupo não é apresentado neste exemplo devido ao grande tamanho da planilha exemplo e de sua semelhança estrutural com o segundo grupo. A quantidade de colunas no segundo e no terceiro grupo de colunas depende da aplicação para a qual é realizado o projeto de teste.

	A	B	C	D	E	F	G	G	G
1	**Caso de Teste**			**Dados de Entrada**					
2	**ID**	**Nome**		**Promotion Id**	**Description**	**Start Date**	**End Date**	**Priority**	**Scope Level**
3	1	Promoção Válida 1		PV1	Promoção Um	12/31/2005	12/31/2015	1	Comb Object
4	2	Promoção Válida 2		PV2	P	03/09/2004	01/01/2015	2	Main Object
5	3	Promoção Válida 3		PV3	Promo tamanho max	01/01/2005		2	Customer
6	4	Promoção Válida 4		PV4	Promoção Um	01/01/2005	01/01/2005	99	Cust Group
7	5	Ident. Inválido – Nulo		NULO					
8	6	Ident. Inválido – Repetido		PV1					
9	7	Descrição Inválida – Nula			NULO				
10	8	Descrição Inválida – Tam			Promo tamanho maior				
11	9	Data Inicial Inválida – Nula				NULO			
12	10	Data Inicial Inválida – Texto				String			
13	11	Data Inicial Inválida – Data				40/20/2003			
14	12	Data Final Inválida – Texto					String		
15	13	Data Final Inválida – Data					40/20/2003		
16	14	Prioridade Inválida – Zero						0	
17	15	Prioridade Inválida – Negativo						-1	

Figura 2 – Exemplo de Planilha Data-Driven

As planilhas *keyword-driven* especificam o procedimento de teste que deve ser seguido durante a execução dos casos de teste. Este procedimento é que controla o fluxo de ações executadas sobre a aplicação em teste. A Figura 3 apresenta um exemplo de uma planilha *keyword-driven*. A primeira coluna contém as palavras-chave que fazem o interpretador disparar a execução de uma ação de um script *data-driven* ou de um script *keyword-driven*. As outras colunas contêm parâmetros para as ações chamadas pelas palavras-chave, sendo que a ordem e significado destes parâmetros variam para cada ação de teste. O calor do parâmetro é apresentado em negrito, logo abaixo da célula que contém o nome do parâmetro. Por exemplo, o comando "Apertar Botão" instrui o interpretador da planilha a disparar o script *keyword-driven* que realiza a ação de apertar um botão em uma janela especificada pelos parâmetros.

3.2. Arquitetura do *Framework* AutoTest

Primeiramente é apresentado na Figura 4 um diagrama que focaliza os componentes básicos do *framework* AutoTest. O núcleo do sistema é o Interpretador, um script central capaz de

interpretar uma seqüência de comandos da planilha *keyword-driven* e tomar as ações correspondentes em tempo de execução. Com isso, não há necessidade de passos intermediários, como a compilação e geração de scripts de teste para os procedimentos de teste. Esta abordagem permite, também, que verificações e decisões mais complexas sejam feitas durante a execução de cada ação, dando flexibilidade e robustez ao sistema.

	A	B	C	D	E	F
1	**Seção**	**1 - Inicia o Módulo de Promoções**				
2		Executável	Parâmetros	Janela	Estado	
3	**Iniciar Aplicação**	**C:\Tst\Dct\run.bat**	**--gui**	**Aplicação de Promoções**	**Maximizado**	
4	**Seção**	**2 - Testa os casos válidos de cadastro de promoções**				
5		Planilha				
6	**Cadastrar Promoções**	**CsTeste1PR FAT inclusão de promoções (dados válidos).xls**				
7	**Seção**	**3 - Tenta cadastrar uma promoção com data inválida (detecção automatizada de erro)**				
8		Janela	Menu	SubMenu 1		
9	**Selecionar Menu**	**Aplicação de Promoções**	**Promoção**	**Criar promoção**		
10		Janela	Botão			
11	**Apertar Botão**	**Promoção**	**Maximizar Janela**			
12		Janela	Campo	Texto		
13	**Editar Campo**	**Promoção**	**Validade Final**	**01012002**		
14		Janela	Teclas			
15	**Digitar**	**Promoção**	**^{F4}**			
16		Abortar planilha?	Operação	Janela	Campo	Texto
17	**Em Caso de Erro**	**Não**	**Editar Campo**	**Promoção**	**Validade Inicial**	

Figura 3 – Exemplo de Planilha Keyword-Driven

O *framework* AutoTest foi desenvolvido sobre a plataforma IBM Rational Functional Tester for Java and Web [7], uma ferramenta de automação de teste que oferece suporte somente à programação e a execução de scripts de teste. Apesar de dar suporte nativo à técnica "*record & playback*" de automação de testes, o Functional Tester usa Java como linguagem dos scripts de teste, permitindo a aplicação de técnicas mais avançadas de programação de scripts – como a técnica *keyword-data-driven* aqui definida.

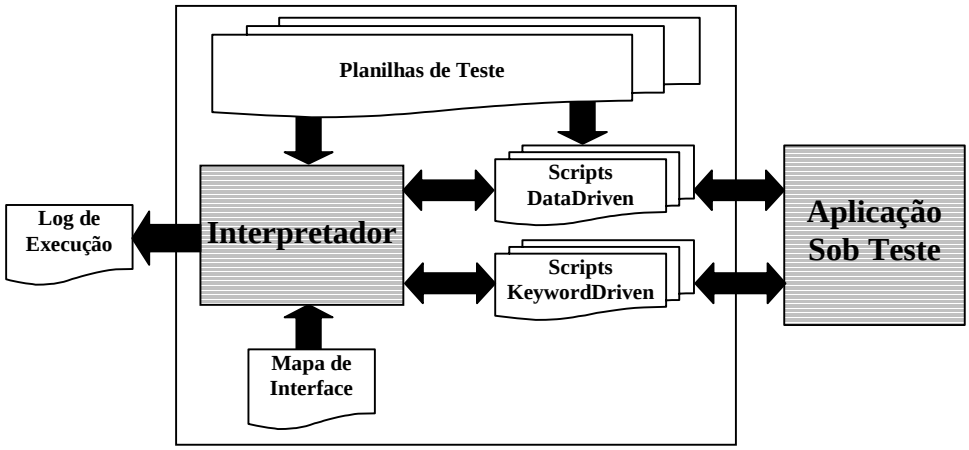

Figura 4 – Componentes Básicos do Framework AutoTest

Os dados de teste e os procedimentos de teste são mantidos nas planilhas eletrônicas data-*driven* e *keyword-driven*, seguindo a técnica *keyword-data-driven* de automação de teste definida na seção anterior. Cada comando existente na planilha *keyword-driven* – representado por meio de uma palavra-chave, dispara a execução de um script *data-driven* ou script *keyword-driven*, os quais por sua vez interagem com a aplicação sob teste. A criação e utilização das palavras-chave reconhecidas pelo Interpretador são feitas de forma dinâmica. Assim pode-se partir de um conjunto inicial de palavras-chave e, se necessário, este conjunto

pode ser facilmente estendido, bastando para isso criar o script de teste *data-driven* ou *keyword-driven* que implemente a execução da ação associada à nova palavra-chave criada.

O Mapa de Interface é um arquivo que contém nomes fictícios dos componentes da interface gráfica (GUI) da aplicação e as propriedades que os identificam unicamente. Este mapa é utilizado para que todo componente da GUI seja referenciado nos scripts e nas planilhas de teste por um nome que independa de mudanças da aplicação, com o objetivo de que as alterações introduzidas na interface gráfica da aplicação impliquem, apenas, em atualizações no Mapa de Interface, preservando e tornando o teste mais robusto.

Todas as ações executadas pelo sistema são registradas em um relatório com todos os parâmetros de teste, as ações executadas, os resultados esperados e obtidos para cada caso de teste, e a data e a hora de execução de cada evento. Além da descrição textual, o relatório apresenta, nas duas primeiras colunas, códigos que identificam a natureza das mensagens (informação, erro, início de caso de teste, etc) para tornar fácil a filtragem do conteúdo do relatório por meio de editores de texto ou a sua conversão para um formato específico.

Este *framework* pode ser utilizado na automação tanto do teste de funcionalidades de processamento interativo (por meio de interface gráfica) quanto no teste de funcionalidades de processamento *batch*. Além dos componentes básicos do *framework* AutoTest, descritos nesta seção, existe um amplo conjunto de componentes adicionais que fazem parte deste *framework*. A Figura 5 apresenta a arquitetura completa do *framework* AutoTest.

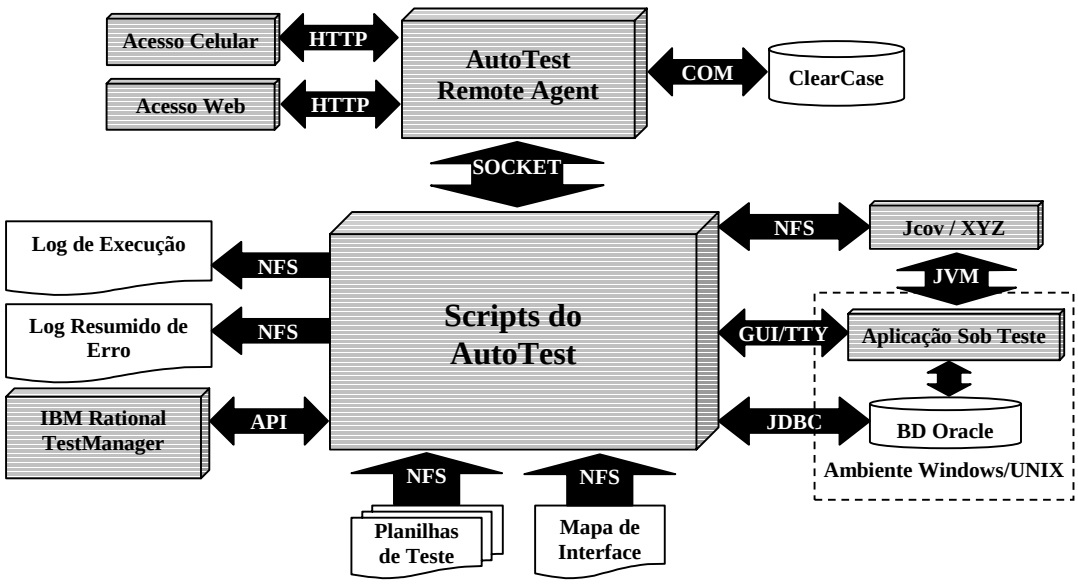

Figura 5 – Arquitetura Completa do Framework AutoTest

A interação dos scripts de teste com os demais componentes do *framework* AutoTest é realizada por meio do uso de tecnologias e protocolos específicos, também apresentados na Figura 5, que fazem parte da definição do *framework*. A seguir é apresentada uma breve descrição de cada um dos componentes adicionais do *framework*:

- IBM Rational Test Manager: uma ferramenta de gerenciamento de projeto de teste que possibilita a integração do *framework* AutoTest com as demais ferramentas da suíte de desenvolvimento da IBM Rational. Seu uso possibilita um melhor gerenciamento de resultados obtidos e a obtenção de relatórios mais específicos e detalhados;

- Log Resumido de Erro: um relatório, em formato texto simples ou HTML, que apresenta apenas um resumo dos casos de teste que foram bem sucedidos e dos que falharam, cujo objetivo é facilitar a análise dos resultados obtidos pelos analistas de teste. Para os casos de teste que falharam é apresentada a mensagem de erro obtida;

- JCov / XYZ: qualquer ferramenta de análise de cobertura de código que se queira utilizar durante a execução do teste. O *framework* AutoTest permite que, na ausência desta, seja utilizada uma opção da máquina virtual Java da Sun para o mesmo fim;

- BDs Oracle: bancos de dados utilizados durante a execução automatizada do teste, incluindo o banco de dados acessado pela aplicação sob teste e outros bancos de dados disponíveis para consultas e verificações complementares que sejam necessárias;

- Remote Agent: uma ferramenta web, acessada por meio de um navegador, que permite a execução e o acompanhamento de testes automatizados, com a facilidade de uma interface interativa. O acesso pode ser realizado via navegador de internet ou via navegador WAP, por meio de um telefone celular.

4. Estudo de Caso: Automação do Teste de um Sistema de Faturamento

Para a validação do *framework* AutoTest foi realizada a aplicação do mesmo na automação de teste de um sistema de faturamento de grande porte, bastante complexo, em desenvolvimento pela empresa CPqD Telecom & IT Solutions, por meio da DSB – Diretoria de Soluções em Billing. Este sistema se aplica a empresas de telecomunicações e é composto por um conjunto de oito módulos responsáveis pela realização de: tarifação, tributação, faturamento, promoções, arrecadação, cobrança, atendimento a cliente e contabilidade.

Este sistema conta atualmente com aproximadamente 250.000 linhas de código. Além de sua alta complexidade, sistemas deste tipo possuem uma necessidade de rápida evolução funcional para acompanhar a constante transformação do setor de telecomunicações. Contando com uma equipe de desenvolvimento formada por cerca de 160 pessoas, das quais cerca de 25 trabalham na atividade de teste, a DSB precisa lidar com uma solução de teste bastante heterogênea. A maioria destes módulos é disponibilizada na arquitetura cliente-servidor, mas alguns deles também são disponibilizados na arquitetura web.

4.1. Metodologia de Coleta de Métricas

Apesar de haver um consenso entre os especialistas dos ganhos que podem ser alcançados com a utilização de uma estratégia de automação de teste de software, as empresas que usufruem desta tecnologia normalmente possuem dificuldades em avaliar o real benefício que está sendo alcançado com o investimento realizado. Visando justificar o investimento aplicado, torna-se de fundamental importância que sejam definidas estratégias de coleta e análise de métricas relacionadas ao processo de teste de software, englobando a atividade de automação de teste. É por meio destas métricas que a execução manual do teste de software pode ser comparada à execução automatizada do mesmo conjunto de casos de teste.

Apesar da grande importância da coleta e análise de métricas, ainda não estão disponíveis na literatura definições sobre os tipos de métricas que devem ser colhidas especificamente em relação à automação de teste. Deste modo, como parte deste estudo de caso, foi realizado um amplo trabalho de definição desses tipos de métricas a serem considerados. Como resultado dessa atividade foi elaborado um *template* de planilha de métricas de teste, o qual passou a fazer parte do *framework* AutoTest. Segue uma breve descrição das métricas a serem coletadas, de acordo com o *template* definido:

1) Manutenção do Projeto de Teste, composto por:
   a) Dados de Teste: tempo total gasto no detalhamento dos dados de teste (dados de entrada e resultados esperados) para um determinado caso de teste;

b) Procedimento de Teste: tempo total gasto no detalhamento dos procedimentos de teste (conjunto de passos e ações sobre a aplicação) para um determinado conjunto de casos de teste;

c) Quantidade de Alterações: quantidade de alterações efetuadas no projeto de teste (dados de teste e procedimento de teste), usada para calcular uma média do tempo gasto na manutenção no projeto de teste.

2) Execução Manual do Teste, composto por:

a) Execução: tempo total gasto na execução manual de um caso de teste e na análise dos resultados obtidos, para saber se ocorreu uma falha ou não;

b) Registro de Erro: tempo total gasto nos registros de erros que podem ser detectados por meio da execução manual de um caso de teste;

c) Quantidade de Execuções: quantidade de execuções manuais realizadas para um caso de teste, usada para calcular a média do tempo gasto na execução manual do caso de teste no registro de erros detectados;

d) Quantidade de Erros: quantidade total de erros que foram detectados durante a execução manual dos casos de teste.

3) Execução Automatizada do Teste, composto por:

a) Execução: tempo total gasto na execução automatizada de um caso de teste e na análise também automatizada dos resultados obtidos, para saber se ocorreu uma falha ou não;

b) Análise do Log de Execução Automatizada: tempo total gasto na análise manual de um log de execução automatizada gerado pelo *framework* AutoTest e que contém a descrição dos casos de teste que foram bem sucedidos e dos casos de teste que falharam, incluindo o desvio funcional ocorrido no caso de falhas;

c) Registro de Erro: tempo total gasto nos registros de erro que podem ser detectados por meio da execução automatizada de um caso de teste;

d) Quantidade de Execuções: quantidade de execuções automatizadas realizadas para um caso de teste, usada para calcular uma média do tempo gasto na execução automatizada do caso de teste no registro de erros detectados;

e) Quantidade de Erros: quantidade total de erros adicionais que foram detectados durante a execução automatizada dos casos de teste.

As primeiras versões do *template* da planilha de métricas foram definidas com uma quantidade bem maior de tipos de métricas a serem coletadas, visando uma maior disponibilidade de dados a serem usados em futuras análises. Entretanto, à medida que as métricas foram sendo coletadas, observou-se que o tempo gasto nesta coleta estava sendo muito grande. Com isso, o escopo de métricas inicial foi revisto a fim de se estabelecer um conjunto mais adequado do ponto de vista da relação custo/benefício.

4.2. Resultados Obtidos

Nesta seção são apresentados os resultados obtidos na aplicação da automação de teste no sistema de faturamento alvo deste estudo de caso e dos benefícios alcançados. Foram considerandos dois módulos do sistema de faturamento, que se encontram em momentos distintos do ciclo de vida: o Módulo de Promoções, em desenvolvimento de sua primeira versão; e o Módulo de Atendimento a Clientes, já implantado em várias empresas operadoras, que sofreu evolução de suas funcionalidades em novas versões. Estes módulos, descritos

brevemente a seguir, representam respectivamente 15% e 20% aproximadamente do tamanho total do sistema de faturamento do qual eles fazem parte.

O Módulo de Promoções é responsável pela implementação das promoções lançadas pelas empresas de telecomunicações com o objetivo de fidelizar seus clientes e cativar novos. Cada promoção é formada por um conjunto de benefícios, que podem ser descontos ou franquias sobre os serviços usados pelo cliente e cobrados em uma conta telefônica. Este módulo possui uma funcionalidade interativa (manutenção de promoções) e uma funcionalidade *batch* (aplicação de promoções).

O Módulo de Atendimento a Clientes é utilizado pela equipe da empresa operadora de telecomunicações no atendimento a clientes que possuem dúvidas ou reclamações sobre serviços ou valores cobrados em sua conta telefônica. As reclamações podem ocasionar alterações a serem refletidas em uma conta futura ou na geração de uma nova conta. Este módulo possui apenas funcionalidades interativas.

4.2.1. Módulo de Promoções

A execução completa do teste automatizado deste módulo contempla: testes de inclusão, alteração e exclusão de promoções, com dados válidos e dados inválidos; e testes de aplicação de promoções. Embora existam diferenças que causam alguns impactos na automação das duas funcionalidades deste módulo, por simplicidade, os resultados apresentados a seguir consideram o conjunto total de casos de teste automatizados para as duas funcionalidades.

Por se tratar de um módulo novo, foi necessário realizar um único projeto de teste, a ser usado em ambas abordagens manual e automatizada, incluindo a elaboração das planilhas de teste. Do total de casos de teste projetados, primeiramente foram executados apenas os casos de teste mais críticos de forma manual. Em paralelo, foram realizadas as adaptações no *framework* AutoTest, preparando-o para a execução automatizada dos casos de teste. Depois que os principais erros foram detectados e corrigidos, e a aplicação se tornou mais estável, foram realizadas as demais execuções dos casos de teste de forma automatizada, porém no escopo completo das planilhas do projeto de teste realizado inicialmente. Durante a execução automatizada, erros adicionais foram detectados, devido ao aumento do escopo de casos de teste executados. A Tabela 1 apresenta um resumo das principais métricas coletadas durante as execuções de teste – tanto manual quanto automatizada – do Módulo de Promoções.

Tabela 1 – Principais Métricas Coletadas para o Módulo de Promoções

Tipo de Métrica	Teste Manual	Teste Automatizado
Número de Casos de Teste Executados	930	1644
Cobertura Funcional dos Casos de Teste[1]	65 %	88 %
Erros Detectados	174	+ 33
Tempo de Projeto de Teste	101 h	101 h
Tempo de uma Execução Completa dos Casos de Teste	123 h	14 h
Análise dos Resultados e Registro de Erros	34 h	28 h

A Figura 6 apresenta um gráfico da relação entre o tempo gasto (em horas) nas execuções manual e automatizada dos casos de teste – destacando o ganho da abordagem automatizada em relação à manual. Para cada abordagem, é considerado primeiramente o tempo gasto com a atividade de projeto de teste, que é o mesmo para ambas. Em seguida são consideradas a primeira execução do teste e as re-execuções subseqüentes. O tempo gasto com a adaptação do *framework* é considerado na primeira execução automatizada.

[1] Porcentagem de comandos da aplicação executados pelo menos uma vez durante a execução dos casos de teste.

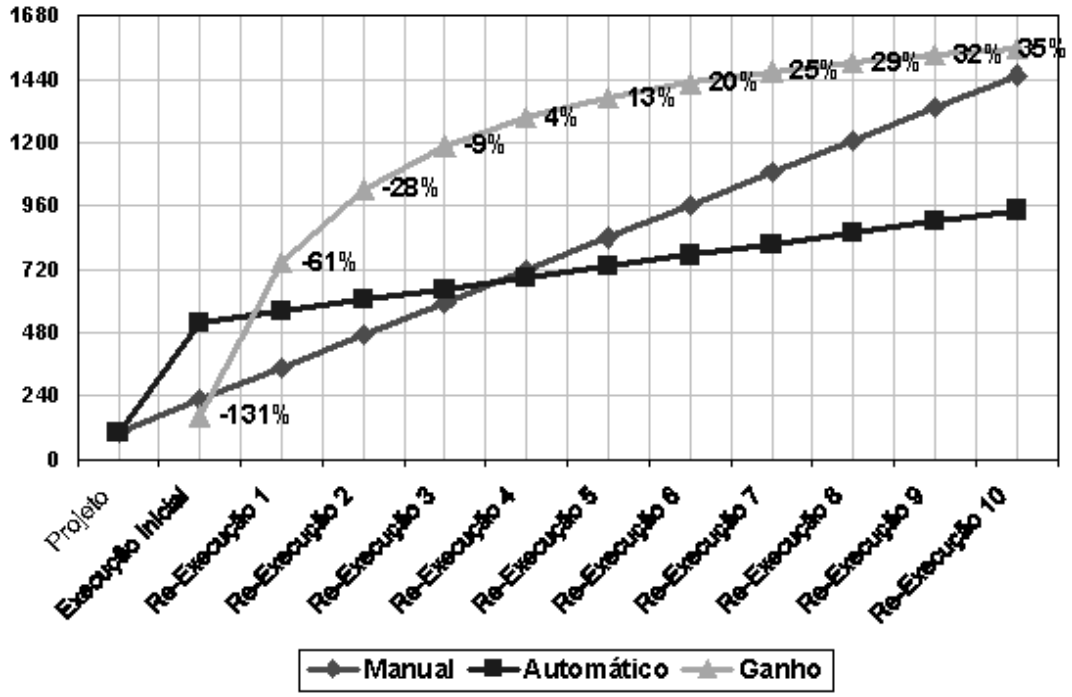

Figura 6 – Teste Manual X Teste Automatizado do Módulo de Promoções

Numa comparação direta entre os tempos de execução dos casos de teste de cada abordagem, verifica-se que a taxa de execução do teste automatizado é de 117 casos de teste por hora contra apenas 7,5 casos de teste por hora do teste manual, uma diferença de aproximadamente 15 vezes. Porém, deve-se considerar também o tempo gasto com a adaptação do *framework* para a realização do teste automatizado, o que inclui a elaboração de novos scripts *data-driven* e de alguns scripts *keyword-driven*. Como este tempo é relativamente alto, o ganho de produtividade do teste automatizado em relação ao manual, para este caso especificamente, é obtido apenas a partir da quarta re-execução do teste, o que é perfeitamente aceitável por se tratar do teste de regressão de um módulo novo.

O ganho de qualidade com a abordagem automatizada, para este módulo, pode ser confirmada pela quantidade de erros detectados adicionalmente na execução automatizada dos casos de teste. Os 33 erros adicionais representam um aumento de 20% dos erros detectados em relação puramente à abordagem manual. Embora o aumento da quantidade de casos de teste nem sempre leve a detecção maior de erros, o que depende da qualidade dos casos de teste, neste caso o escopo maior do teste automatizado justifica esse aumento.

4.2.2. Módulo de Atendimento a Clientes

A execução completa do teste automatizado deste módulo contempla testes de reclamação e retificação de contas telefônicas e de cancelamento de reclamações e retificações realizadas. Por motivos de simplicidade, a apresentação dos resultados obtidos também considera o conjunto total de casos de teste automatizados para as quatro funcionalidades do módulo.

Por se tratar de um módulo em evolução, as planilhas de teste não foram criadas durante a atividade de projeto de teste, mas apenas adaptadas a partir do projeto anterior para cobrir as funcionalidades adicionadas ou alteradas. Como o impacto da evolução do módulo nas planilhas de teste foi pequena, não foi necessário realizar a primeira execução do teste de forma manual. Além disso, poucas adaptações precisaram ser feitas no *framework* AutoTest.

A Tabela 2 apresenta um resumo das principais métricas coletadas durante as execuções de teste – tanto manual quanto automatizada – do Módulo de Atendimento a Clientes.

Tabela 2 – Principais Métricas Coletadas para o Módulo de Atendimento a Clientes

Tipo de Métrica	Teste Manual	Teste Automatizado
Número de Casos de Teste Executados	-	246
Cobertura Funcional dos Casos de Teste	-	71 %
Erros Detectados	-	18
Tempo de Projeto de Teste	-	718 h
Tempo de uma Execução Completa dos Casos de Teste	525 h (estimado)	4 h
Análise dos Resultados e Registro de Erros	100 h (estimado)	62 h

Como o teste manual não foi executado, algumas métricas tiveram que ser estimadas para serem usadas na comparação de abordagens, com base em execuções já realizadas para este mesmo módulo. Nota-se uma grande diferença entre os tempos de uma execução completa dos casos de teste: 525 horas estimadas para o teste manual e apenas 4 horas medidas para o teste automatizado. Esta diferença se justifica por se tratar de um módulo com uma grande quantidade de operações que demandam muitos acessos ao banco de dados, via SQL, tanto para preparar os testes quanto para analisar resultados.

A Figura 7 apresenta um gráfico, com os resultados do Módulo Atendimento a Clientes, similar ao apresentado na Figura 6. Para cada abordagem é considerado o tempo gasto com evolução do projeto de teste primeiramente. Em seguida são consideradas as re-execuções subseqüentes, dado que não foi considerada uma primeira execução por se tratar de uma evolução em um módulo já previamente testado automaticamente. O tempo gasto com a adaptação do *framework* é considerado na evolução do projeto de teste automatizado.

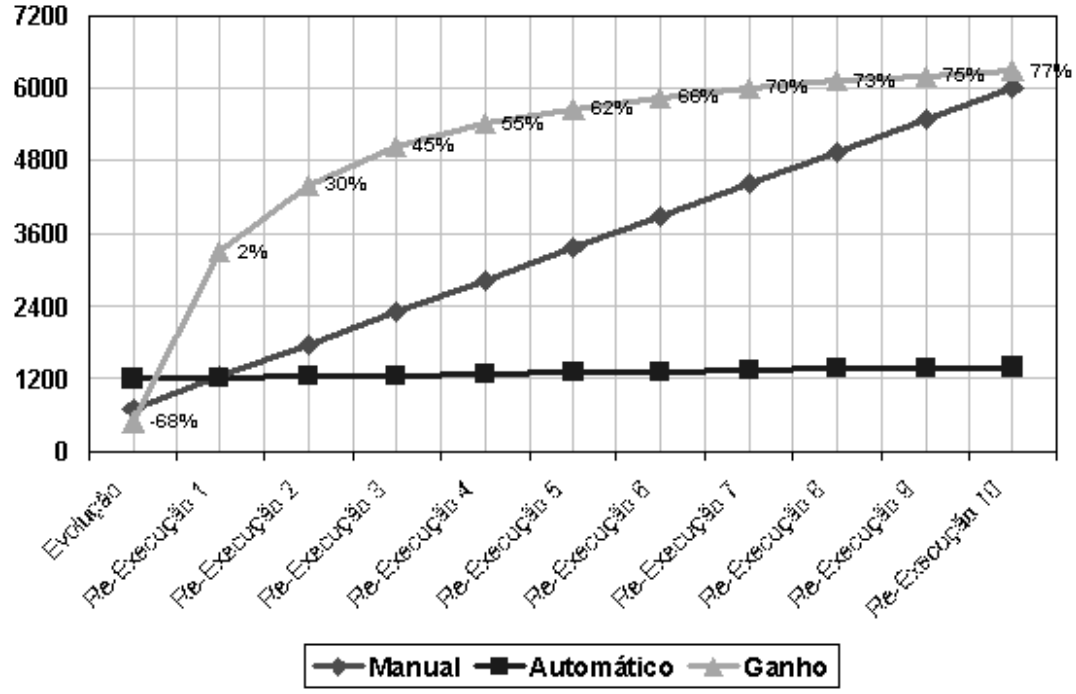

Figura 7 – Teste Manual X Teste Automatizado do Módulo de Atendimento a Clientes

Pelos dados apresentados, verifica-se que expressivos resultados foram alcançados com a automação de testes do Módulo Atendimento a Clientes, visto que desde a primeira re-execução do teste, a execução automatizada ganharia de uma re-execução manual. Esse

resultado foi conseguido por se tratar de um teste de regressão de um módulo já existente e que foi evoluído para uma nova versão, o tipo de teste que oferece o maior potencial para a obtenção de melhores resultados.

Considerando que já foram, até a finalização deste estudo de caso, realizadas 7 re-execuções de teste automatizadas, foi obtido um ganho de tempo de, aproximadamente, 70% em relação a um possível teste manual. Infelizmente, o ganho de qualidade, neste caso, é praticamente imensurável, dado que não é possível afirmar quantas re-execuções de teste manuais teriam sido feitas e nem o escopo de tais re-execuções.

4.3. Análise dos Resultados

Por meio da realização deste estudo de caso, demonstrarem-se os ganhos da abordagem de teste automatizada em relação à abordagem manual para as funcionalidades alvo do sistema escolhido. De uma forma geral, os resultados obtidos se apresentaram muito satisfatórios, superando em alguns pontos as expectativas. O *framework* se mostrou robusto, expansível e extensível o suficiente para permitir sua reutilização na automação de teste de várias funcionalidades do sistema em questão.

A aplicação do *framework* AutoTest possibilitou a obtenção de uma grande economia de esforço e tempo necessários para a realização do teste, ganhando-se eficiência e confiança nos resultados do teste – principalmente na aplicação de teste de regressão. Outro ponto a ser enfatizado, em termos de qualidade dos testes automatizados, é a aplicação total dos testes em todas as re-execuções, enquanto que a do teste manual tende a se restringir a um subconjunto a cada nova re-execução por uma série de fatores humanos e priorizações gerenciais. Como o simples aumento na quantidade de casos de teste não garante a melhoria na qualidade do teste, a automação de teste deve ser acompanhada de técnicas que visem a identificação de casos de teste com maior possibilidade de detecção de erros.

A comparação das abordagens de teste manual e automatizada se mostrou limitada em alguns casos. Como, depois que o *framework* para automação de teste é estabelecido para um determinado projeto, a execução manual normalmente não é mais realizada, não há a coleta de métricas para a execução manual. Assim, a comparação das demais re-execuções para as duas abordagens de teste pode ser apenas estimada. Não há muitas garantias, apesar das fortes indicações, de que esta estimativa leve ao real ganho na qualidade do produto, por ser uma medida mais subjetiva que a produtividade. Uma forma de melhor avaliá-la seria a comparação da satisfação do cliente antes e depois da implantação de automação de teste, não realizada neste estudo de caso pela ausência de histórico formal das reclamações do cliente.

5. Trabalhos Relacionados

Existem várias ferramentas de suporte à automação de teste de software, porém estas ferramentas apresentam escopo limitado de funcionalidades, as quais são direcionadas a determinadas ações de automação de teste. Diferentemente do *framework* AutoTest, a aplicação pura destas ferramentas não oferece garantias de obtenção de melhoria na produtividade e na qualidade de software, aliados a um baixo custo de automação.

Uma grande parte das ferramentas de automação de teste disponíveis comercialmente é fortemente voltada para a técnica *record & playback*, cujas desvantagens e limitações já foram apresentadas em sua breve descrição neste artigo. As ferramentas QA Wizard [20] e e-TEST [21] podem ser utilizadas exclusivamente com esta técnica. Já as ferramentas Functional Tester [7] e Robot [7], WinRunner [22] e QuickTestPro [22], TestSmith [23], QARun [24], e SilkTest [25], além da funcionalidade de gravação de scripts de teste,

oferecem uma linguagem de programação permitindo que a técnica de programação de scripts seja utilizada. Nenhuma destas ferramentas oferece suporte nativo às técnicas de automação de teste *data-driven* e *keyword-driven*, não trabalhando diretamente, por exemplo, com o conceito de mapa de interface. Apesar disso, algumas dessas ferramentas oferecem certa facilidade, em diferentes níveis, no uso da técnica *data-driven*.

O *framework* AutoTest um conjunto maior de suporte a automação de teste do que estas ferramentas. Na realidade, uma delas – no caso a ferramenta IBM Rational Functional Tester for Java and Web [7] – é um dos componentes deste *framework*. A idéia do *framework* é oferecer uma ambiente reutilizável para o especialista em automação de teste trabalhar, ao invés dele ter que simplesmente começar seu trabalho sempre do ponto inicial, tendo que programar todos os scripts necessários e definir os padrões a serem utilizados.

Outras duas ferramentas de suporte a automação de teste são: TestMentor [26] e JFunc [27], uma extensão funcional da ferramenta JUnit usada para a automação do teste de unidade. Ambas oferecem suporte a teste funcional por meio da criação de classes de teste que devem interagir diretamente com as classes do sistema, sendo compiladas e ligadas juntas. Seu conceito de teste funcional não é equivalente ao usado no escopo das ferramentas descritas anteriormente e do *framework* AutoTest, que é testar o sistema pronto como um todo, do ponto de vista do usuário, por meio da interface gráfica disponibilizada para isso.

6. Conclusão e Trabalhos Futuros

Embora a automação de teste de software seja vista como uma forma de melhorar a produtividade e a qualidade de software, ela não é indicada para o teste de qualquer tipo de funcionalidade. As funcionalidades mais propícias para o uso de tal abordagem são aquelas que envolvem a execução de tarefas repetitivas e cansativas, facilmente suscetíveis a erros, ou impossíveis de serem realizadas manualmente. Além disso, existem várias técnicas que podem ser utilizadas na automação do teste de uma funcionalidade, as quais possuem vantagens e desvantagens dependendo da natureza da funcionalidade em questão.

Assim, o ganho com a automação depende fortemente de sua implantação sistemática. Conseqüentemente, esta atividade apresenta as mesmas características comuns a outros projetos de software, sendo necessários planejamento, análise, projeto, implementação e até mesmo teste. Segundo [8], um trabalho mínimo de criação, manutenção e documentação de testes automatizados é, em média, de 3 a 10 vezes mais longo que o mesmo trabalho manual.

Este artigo apresentou a definição de um *framework* reutilizável para automação de teste funcional que visa facilitar a aplicação do teste de software automatizado, de modo a se obter maiores ganhos em relação a uma abordagem de teste puramente manual. Este *framework*, chamado AutoTest, é baseado principalmente na técnica *keyword-data-driven*, uma técnica de automação de teste definida com base em outras duas técnicas cuja aplicação conjunta apresenta melhores benefícios. A aplicação do *framework* foi realizada por meio de um estudo de caso também apresentado neste artigo.

Como trabalho futuro pretende-se estender o *framework* AutoTest para contemplar ainda mais a automação de teste de sistemas baseados na plataforma web. Será necessária a realização de estudos para se identificar quais as características que um sistema web possui e que causam impacto na estratégia de automação de teste suportada pelo *framework* apresentado. Além disso, pretende-se também avaliar a viabilidade de extensão do *framework* para suportar os testes estruturais, ou seja, testes de caixa-branca.

Agradecimentos

À FUNTTEL – Fundo para o Desenvolvimento Tecnológico das Telecomunicações que, por meio da Fundação CPqD, investiu na realização desta pesquisa como apoio ao desenvolvimento de tecnologia de software nacional com alta qualidade e produtividade.

Referências

1. Bach, J., *"Test Automation Snake Oil"*, 14th International Conference and Exposition on Testing Computer Software, 1999.
2. Binder, R. V., *"Testing Object-Oriented Systems – Models, Patterns, and Tools"*, Addison-Wesley,1999.
3. Dustin, E., *"Lessons in Test Automation"*. STQE – The Software Testing & Quality Engineering Magazine, 1999.
4. Fewster, M. & Graham, D., *"Software Test Automation"*, Addison-Wesley, 1999.
5. Fewster, M., *"Common Mistakes in Test Automation"*, Proceedings of Fall Test Automation Conference, 2001.
6. Hendrickson, E., *"The Differences Between Test Automation Success And Failure"*, Proceedings of STAR West, 1998.
7. IBM Rational. Disponível em: http://www-306.ibm.com/software/rational/.
8. Kaner, C., *"Improving the Maintainability of Automated Test Suites"*, Proceedings of the Thenth International Quality Week, 1997.
9. Kaner, C., *"Architectures of Test Automation"*, Proceedings of Los Altos Workshops on Software Testing, 2000.
10. Kit, E., *"Integrated, Effective Test Design and Automation"*, Software Development Magazine, fevereiro/1999.
11. Marick, B., *"Classic Testing Mistakes"*, Proc. of STAR Conference, 1997.
12. Myers, G., *"The Art of Software Testing"*, John Wiley & Sons, 1979.
13. Nagle, C., *"Test Automation Frameworks"*, disponível em http://members.aol.com /sascanagl/Data DrivenTestAutomationFrameworks.htm, 2000.
14. Pettichord, B., *"Seven Steps to Test Automation Success"*, Proc. of STAR West, 1999.
15. Pettichord, B., *"Capture Replay - A Foolish Test Strategy"*, Proc. of STAR West, 2000.
16. Pressman, R., *"Engenharia de Software"*, Makron Books do Brasil, 1992.
17. Rocha, A. R. C., Maldonado, J. C. & Weber, K. C., *"Qualidade de software – Teoria e prática"*, Prentice Hall, 2001.
18. Tervo, B., *"Standards For Test Automation"*, Proc. of STAR East, 2001.
19. Zambelich, K., *"Totally Data-driven Automated Testing"*, disponível em http://www.sqa-test.com/w_ paper1.html, 1998.
20. QA Wizard. Disponíel em: http://www.seapine.com/qawizard.html.
21. e-TEST suíte. Disponível em: http://www.empirix.com.
22. Mercury Interactive. Disponível em http://www.mercuryinteractive.com/
23. TestSmith. Disponível em: http://qualityforge.com/testsmith/index.html.
24. QARun. Disponível em: http://www.compuware.com/products/qacenter/qarun.htm.
25. SilkTest. Disponível em: http://www.segue.com/products/functional-regressional-testing/ silktest.asp.
26. SilverMark's Test Mentor – Java Edition. Disponível em: http://www.javatesting.com/ Product/java/stm/index.html.
27. JFunc: JUnit Functional Testing Extension. Disponível em: http://jfunc.sourceforge.net.