UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Framework para automação de testes funcionais utilizando o *Rational Functional Tester*

TRABALHO DE GRADUAÇÃO

Aluno: Rafael Oliveira Nóbrega (ron@cin.ufpe.br)
Orientador: Alexandre Marcos Lins de Vasconcelos (amlv@cin.ufpe.br)

Recife, 10 de Outubro de 2006.

Assinatura

Este Trabalho de Graduação é resultado dos esforços do aluno Rafael Oliveira Nóbrega, sob a orientação do professor Alexandre Marcos Lins de Vasconcelos, sob o título de "*Framework para automação de testes funcionais utilizando o Rational Functional Tester*". Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

_____

Rafael Oliveira Nóbrega

_____

Alexandre Marcos Lins de Vasconcelos

Agradecimentos

Gostaria de agradecer a todas as pessoas que ajudaram direta ou indiretamente para a realização deste trabalho:

- Agradeço a toda minha família por estar mais uma vez ao meu lado e por sempre me incentivar apesar dos meus momentos de ausência.
- Ao professor Alexandre Vasconcelos pela seriedade da orientação do meu trabalho de graduação.
- Ao pessoal da Qualiti e do C.E.S.A.R., Juliana Lima, Paulo Santos, Eduardo Ferraz, Vinicius Samico e Breno Santos por terem tido bastante paciência comigo ultimamente.
- Aos meus amigos e companheiros de faculdade e meus sócios da Effektiv com os quais vivi momentos inesquecíveis que levarei comigo pra sempre.
- A todos que trabalharam no projeto, Ricardo, Livar, Marilia, Juliana, Fábio, Egito, Diogo, Lenildo e João Marcelo.
- A todos meus amigos da qualiti, por me darem incentivo e conselhos nos momentos decisivos: Paulinho, Maíra, Aninha, Isabela, Rico e Marília.
- A todos meus amigos que entenderam meus momentos de ausência: Gaúcho, Bode, Rico, Bruno, Henrique, Teteu e Noggy.
- E um agradecimento especial à Lila, minha namorada, que sempre entendeu e me apoiou com muita paciência, ternura e carinho em todos meus momentos de ausência.

"Nada vem de graça"

Rafael Nóbrega

Resumo

Este trabalho propõe a construção de um *framework* para automação de testes funcionais utilizando a ferramenta Rational Functional Tester e a técnica para construção e organização de scripts de teste por decomposição funcional.

A utilização de *frameworks* para automatizar testes de software isola a aplicação sob teste dos scripts de teste provendo um conjunto de funções e bibliotecas de funções que serão utilizados durante a automação.

Esta abordagem aumenta a independência na execução e o reuso de código entre os scripts de automação, melhorando a robustez da execução e a manutenibilidade dos scripts, respectivamente.

Sumário

1. Introdução 1
1.1. Metodologia para realização do trabalho 2
1.2. Estrutura do Trabalho 3
2. Automação de Testes Funcionais 4
2.1. Dificuldades da automação de testes de software 4
2.2. Fatores de sucesso da automação de testes de software 5
2.3. Considerações Finais 8
3. Ferramenta IBM Rational Functional Tester 9
3.1. Visão Geral 9
3.2. Processo de Automação 10
3.3. Extensão do Rational Functional Tester 16
3.4. Considerações Finais 19
4. *Framework* para automação de testes funcionais 20
4.1. Melhorias no processo 20
4.2. Arquitetura do Sistema 26
4.3. Considerações Finais 28
5. Resultados Obtidos 30
5.1. Contexto 30
5.2. Análise dos Resultados 31
5.3. Considerações Finais 37
6. Conclusões e Trabalhos Futuros 39
6.1. Dificuldades Encontradas 39
6.2. Trabalhos Futuros 40
Referências 41
APÊNDICE A - Comparação entre Ferramentas de Automação de Testes Funcionais 43

Índice de Figuras

Figura 1 – Rational Software Development Platform - Rational Functional Tester 9
Figura 2 – Tela Inicial do Rational Functional Tester..... 10
Figura 3 –Processo *Record & Playback*..... 11
Figura 4 – Datapool do Rational Functional Tester. 12
Figura 5 – Janela mostrada durante filmagem dos scripts. 13
Figura 6 – Janela mostrada durante execução dos scripts. 13
Figura 7 – Log gerado pela Execução. 15
Figura 8 – Extensão do Processo *Record & Playback*..... 16
Figura 9 – Janelas mostradas durante a adição de um ponto de verificação.. 18
Figura 10 – Processo de Decomposição Funcional 25
Figura 11 – Arquitetura do Sistema 26
Figura 12 – Arquitetura do pacote acoes 27
Figura 13 – Arquitetura do pacote POI 28
Figura 14 – Comparação dos custos de execução automática e manual por ciclo 37

Índice de Tabelas

Tabela 1 – Custos Fixos da Automação de Testes 33
Tabela 2 – Custos Variáveis da Automação de Testes 33
Tabela 3 – Custos Variáveis dos Testes Manuais 34
Tabela 3 – Variáveis do ROI da automação de testes 36
Tabela 4 – Matriz de comparação entre Ferramentas de Automação de Testes Funcionais 46

1. Introdução

A atividade de teste envolve a execução de uma implementação do software com os dados de teste e a análise de suas saídas e de seu comportamento operacional, a fim de verificar se foi executada conforme o esperado [14].

O custo para correção de um erro na fase de manutenção é de sessenta a cem vezes maior do que corrigi-lo durante o desenvolvimento [11].

Dessa forma, a atividade de testes é uma etapa crítica para o desenvolvimento de um software. Embora durante todo o processo de desenvolvimento de software sejam utilizados métodos, técnicas e ferramentas a fim de evitar que erros sejam introduzidos no produto, a atividade de teste continua sendo de fundamental importância para a eliminação dos erros que persistem.

Para se desenvolver um software podemos fazer diferentes tipos de teste, com diversos objetivos - performance, carga, funcionalidade, estresse, entre outros [7].

Os testes funcionais são testes derivados da especificação do software ou componente, ou aqueles que o testador se preocupa somente com a funcionalidade do sistema e não com sua implementação, são os mais utilizados devido à necessidade que os softwares produzidos façam o que foi acordado com o cliente na fase de análise de requisitos [14].

Já a execução de testes de software pode ser feita tanto de forma manual quanto automatizada. A execução manual consiste na reprodução por uma pessoa do teste previamente definido e documento. Já a execução automática consiste na automação do processo de teste manual atualmente em uso [17]. Scripts de teste devem ser construídos para reproduzir os testes que serão executados automaticamente.

Cada uma destas formas tem suas vantagens e desvantagens, mas os testes manuais têm um grande problema em relação ao tempo de execução em relação ao tempo de execução de testes automatizados.

Os testes funcionais devem ser automatizados quando são muito repetitivos e demandam um esforço considerável de tempo quando realizados manualmente [17]. A realização de testes automatizados, além de possibilitar a redução do ciclo de testes, permite um aumento indireto da cobertura do software e, conseqüentemente, da sua qualidade, porque permite que os testadores foquem seus esforços em outros tipos de teste ou em testes que não possam ser automatizados.

Atualmente, existem diversas ferramentas de automação de testes funcionais que vendem a automação de testes como apenas uma gravação da execução de testes manuais e a sua posterior execução repetidas vezes. Infelizmente, isto não funciona na prática. Testes automatizados desta maneira praticamente não utilizam reuso de código o que gera um enorme esforço de manutenção que inviabiliza a grande maioria dos projetos de automação realizados utilizando essa metodologia.

A proposta apresentada neste documento é o desenvolvimento de um *framework* para automação de testes funcionais utilizando uma técnica de automação de testes e a demonstração dos resultados obtidos na utilização deste em uma empresa desenvolvedora de software.

O *framework* sugerido permite que testes funcionais sejam automatizados de maneira enxuta e modular, facilitando a manutenibilidade dos testes automáticos e aumentando o desempenho da execução dos testes.

1.1. Metodologia para realização do trabalho

Esta seção descreve a metodologia empregada para o desenvolvimento deste trabalho. Basicamente, a realização do trabalho foi dividida nas seguintes etapas:

Etapa de Estudo Inicial

- Estudo sobre ferramentas de automação de testes funcionais;
- Estudo sobre metodologias e técnicas de automação de testes;

- Definição do escopo do trabalho.

Etapa de Definição do *Framework*

- Definição da ferramenta de automação de testes
- Definição das funcionalidades da Ferramenta a serem melhoradas;
- Definição das tecnologias utilizadas na implementação do *framework*;
- Implementação das funcionalidades relacionadas ao *framework*, ou seja, dos módulos relacionados à definição e criação de técnicas a serem aplicadas ao processo de automação de testes;

Etapa de Documentação do Trabalho

- Redação final do trabalho de graduação
- Concepção da apresentação do trabalho.

1.2. Estrutura do Trabalho

Além desta introdução, esta monografia está dividida em mais cinco capítulos.

O capítulo 2 apresenta os conceitos sobre automação de testes, suas principais dificuldades e, abordagens e técnicas para superá-las.

O capítulo 3 descreve a ferramenta de automação de testes funcionais IBM Rational Functional Tester, utilizada como base para criação do *framework*.

O capítulo 4 apresenta o *framework* para automação de testes funcionais que utiliza a técnica de decomposição funcional, o qual vai criar scripts que modelam a regra de negócio em funções e bibliotecas utilizadas pelos scripts de teste.

O capítulo 5 apresenta os resultados da utilização do *framework* em uma fábrica de software.

O capítulo 6 apresenta as conclusões e tendências futuras quanto à utilização do *framework* de automação proposto neste trabalho.

2. Automação de Testes Funcionais

Segundo Graham & Fewster em [1], automação de testes não é uma tarefa que trará maior qualidade aos testes. Quando um teste é automatizado, ele terá a mesma eficiência em descobrir erros do que quando executado manualmente. Se o teste em si não acha nenhum erro, a automação deste teste também não vai achar nada, porém de maneira mais rápida.

Depois de feito, um teste automatizado é mais barato do que um manual, o esforço para executá-lo automaticamente é uma pequena fração do esforço de executá-lo manualmente. Porém, o custo de criar e mantê-los é muito mais alto que os testes manuais, levando em média dez vezes mais tempo do que um teste manual [10]. Quanto melhor for a abordagem de automação de testes mais barato será a sua implementação e manutenção.

A qualidade da automação será determinada pela habilidade do profissional automatizador de testes que vai determinar o quão fácil será para adicionar novos testes automatizados, para mantê-los e quais benefícios a automação trará para o produto final.

De acordo com o mito popular, pessoas com pouca experiência de programação podem usar ferramentas de automação de testes e criar rapidamente suítes extensivas de teste. As ferramentas seriam fáceis de usar e a manutenção das suítes de teste não seria um grande problema. Desta forma, um gerente poderia poupar muito dinheiro usando uma destas ferramentas para substituir alguns testadores. Estes mitos são difundidos por vendedores de ferramentas e executivos que não entendem de automação de testes. Porém, a automação de teste não é tão simples assim.

2.1. Dificuldades da automação de testes de software

De acordo com Cem Kaner em [6], existem alguns problemas em relação a esta visão sobre automação de testes de software:

1. Automatizar não é barato

a. Geralmente se leva entre três e dez vezes mais tempo para criar, verificar e documentar minimamente os testes automatizados do que criar e executar uma vez o teste à mão. Os testes que se executa apenas uma ou duas vezes não devem ser automatizados.

2. Esta abordagem cria riscos de novos custos
    a. O custo de encontrar e de reparar erros aumenta ao passar do tempo. Se for gasto muito tempo escrevendo scripts de teste pode se atrasar a execução dos testes para o final do desenvolvimento, quando os erros são mais caros.
3. Estes testes não são poderosos
    a. Os testes que são automatizados são testes que já passaram pelo sistema. Os principais erros são aqueles encontrados durante a criação dos casos de teste, mas eles são normalmente testes manuais, não relacionados aos testes automatizados.
4. Na prática, muitos grupos de teste automatizam apenas os testes simples.

2.2. Fatores de sucesso da automação de testes de software

No mesmo artigo [6], Kaner cita estratégias para uma automação de testes de software de sucesso:

2. Molde a expectativa da gerência em relação ao tempo dos benefícios da automação.
    a. Há um grande benefício em automatizar uma suíte de testes: executar estes testes cinqüenta ou cem vezes durante o desenvolvimento de um *release*. Mesmo se levar dez vezes mais tempo para desenvolver cada teste do que executar cada teste à mão, e mais dez vezes o tempo para a manutenção, haverá ainda um ganho de tempo entre trinta e oitenta execuções manuais.
    b. Provavelmente, o benefício da automação não será percebido no primeiro *release* em que é iniciada a automação. A diminuição do esforço se dará paulatinamente nos próximos *releases*.

3. Desenvolvimento de uma automação de testes é um desenvolvimento de software.
    a. Não se pode desenvolver grandes suítes de teste que sobreviverão e serão úteis em diversos *releases* e que terão baixo custo de manutenção sem um planejamento claro e realístico.
    b. A automação utiliza uma linguagem de programação, mesmo que simplificada, como todo desenvolvimento de software.
    c. Cada caso de teste pode ser encarado como uma funcionalidade.
    d. Os automatizadores, da mesma forma que os desenvolvedores de software fazem, devem: entender os requisitos, entender de programação, adotar uma arquitetura robusta que permita desenvolver, integrar e manter as funcionalidades, entre outras atribuições de qualquer desenvolvedor de software.
4. Use uma arquitetura de testes orientada a dados.
    a. A técnica de testes orientados a dados armazena as entradas dos testes em arquivos separados dos scripts de teste. Quando os scripts de teste são executados, esses dados são lidos destes arquivos ao invés de estarem dentro do código do script. Esta separação clara aumenta consideravelmente a manutenibilidade dos scripts de teste.
5. Use uma arquitetura baseada em um *framework*
    a. Um *framework* provê uma abordagem diferente à automação de testes e geralmente é utilizada com um ou mais estratégias de testes orientados a dados.
    b. Um *framework* isola a aplicação sob teste dos scripts de teste provendo um conjunto de funções em bibliotecas de funções. Os scripts de teste utilizam essas funções como se fossem comandos básicos da linguagem da ferramenta de automação. Então é possível programar os scripts de caso de teste independentemente da interface do usuário.
6. Lembre-se da Realidade da sua equipe
    a. *Frameworks* mal projetados podem destruir o projeto.

b. Muitos testadores excelentes não têm experiência com programação. Eles são imprescindíveis para um esforço de teste, mas não para escrever códigos de automação.
c. Não-programadores podem ser beneficiados por uma abordagem dirigida a dados, pois eles podem desenvolver casos de testes apenas preenchendo tabelas.

7. Considere utilizar outros tipos de automação além da GUI
    a. Automação de Regressão através da GUI pode trazer uma falsa idéia de cobertura que não existe. E isto pode causar uma sobrecarga na equipe deixando os profissionais mais experientes criando e mantendo scripts ao invés de encontrando *bugs.*
    b. Ferramentas de automação de testes pela interface com usuário são bastante úteis, mas requerem um investimento significativo, um planejamento detalhado, uma equipe bem treinada e bastante cuidado.

Outro fator crítico para o sucesso da automação é a seleção da ferramenta adequada. Investir dinheiro, treinamento e tempo na ferramenta errada levará, certamente, ao fracasso automação de testes. De acordo com Hendrickson em [2], para fazer a escolha certa é necessário estabelecer claramente os requisitos para a ferramenta, discutindo com os *stakeholders* do projeto questões como:

1. Quem vai usar a ferramenta, qual o seu propósito e quais problemas essa ferramenta vai nos ajudar a resolver?
2. Que tipo de processo a ferramenta necessita dar suporte e, havendo mudanças no processo, a ferramenta pode facilmente ser adequada?
3. Que funcionalidades a ferramenta precisa ter; que relatórios devem ser extraídos?
4. Quem deve ter acesso à ferramenta e que nível de controle de acesso e administração é necessário?
5. Que tipo de interface com o usuário é necessária? GUI, linha de comando?
6. Com quais plataformas a ferramenta precisa ser compatível? Com que outras ferramentas ela deverá se integrar?

7. Qual o orçamento e tempo disponíveis para aquisição e manutenção?

Independente de qual seja a decisão, adquirir uma ferramenta ou desenvolver seu próprio *framework*, é necessário considerar que esforço e investimento serão necessários no processo de seleção e implantação ou construção da ferramenta antes de iniciar a automação de testes propriamente dita.

2.3. Considerações Finais

A automação de testes precisa ser bem analisada e pensada sobre a sua necessidade e como ela será mais bem adaptada às necessidades de cada processo de desenvolvimento de software. Este capítulo apresentou as principais questões por trás da automação de testes.

Seguindo estas estratégias como orientação geral, este trabalho pretende utilizar uma ferramenta de automação de testes funcionais e criar um *framework* de automação de testes funcionais orientado a dados, para aumentar a produtividade da atividade de testes, e conseqüentemente do software em desenvolvimento.

O apêndice A apresenta um comparativo entre ferramentas de automação de testes funcionais a partir de certos critérios a fim de se avaliar as principais ferramentas do mercado.

3. Ferramenta IBM Rational Functional Tester

O framework de automação de testes funcionais proposto neste trabalho foi desenvolvido em cima da plataforma IBM Rational Functional Tester, ou RFT. Este capítulo se propõe a mostrar a ferramenta RFT, utilizando o processo *Record & Playback* como guia para a apresentação da ferramenta.

Figura 1 – Rational Software Development Platform - Rational Functional Tester

Quando se utiliza automação de testes pela primeira vez, em geral, o primeiro impulso é de utilizar um processo *Record & Playback* que consiste em gravar a utilização da aplicação sob teste e reproduzir a gravação diversas vezes quanto necessário para averiguar a corretude da aplicação [1].

O IBM Rational Functional Tester dá suporte a este processo através de funcionalidades que serão explicadas ao longo deste capítulo.

3.1. Visão Geral

IBM Rational Functional Tester é uma ferramenta de testes automatizados orientada a objeto onde é possível rapidamente gerar scripts gravando testes em uma

aplicação – Java, .NET ou *Web* – e avaliar propriedades ou valores dos objetos da interface do usuário [8].

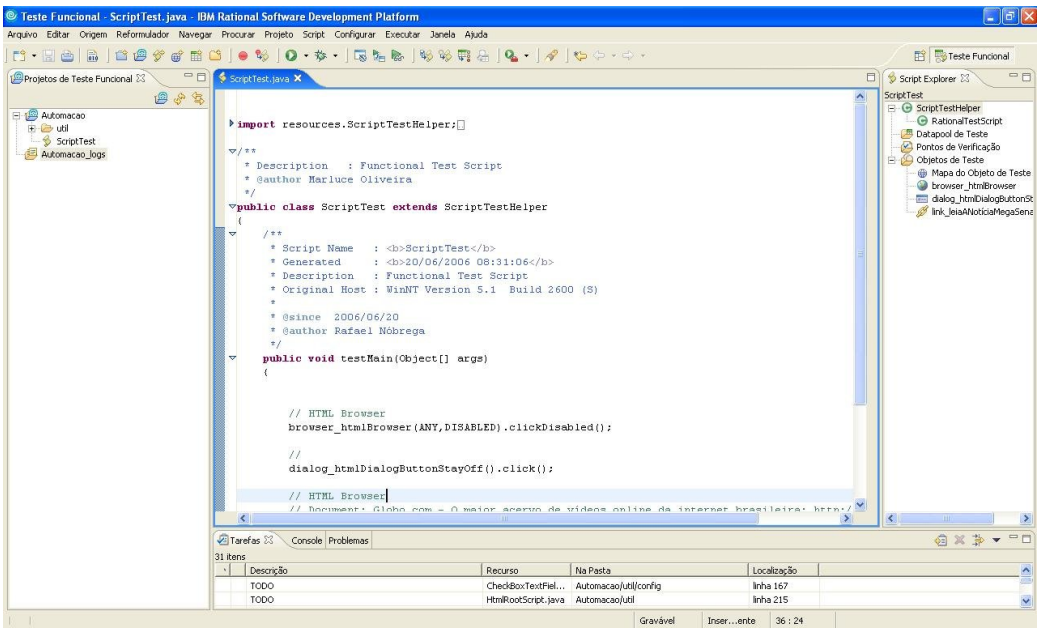

Figura 2 – Tela Inicial do Rational Functional Tester

O Rational Functional Tester oferece uma escolha do ambiente de desenvolvimento e da linguagem de codificação - Java para plataforma *Eclipse* ou *Microsoft Visual Basic .NET* no ambiente *Microsoft Visual Studio .NET*.

Para fins didáticos, vamos apenas apresentar o IBM Rational Functional Testes utilizando a linguagem Java na plataforma Eclipse. Porém, todas as funcionalidades apresentadas também são encontradas no ambiente Visual Studio .Net.

3.2. Processo de Automação

Todo processo de automação de testes manuais pode ser dividido em três grandes tarefas comuns a qualquer processo:

- Gravação do Script de Teste
- Execução do Script de Teste
- Reportagem da Execução Automática

Dentro de cada macro-tarefa acima existem diversas subtarefas que vão realizar trabalhos específicos. Após a realização de todas as subtarefas, a tarefa mais geral pode ser considerada finalizada.

A ferramenta Rational Funcional Tester pode ser utilizada seguindo o seguinte processo:

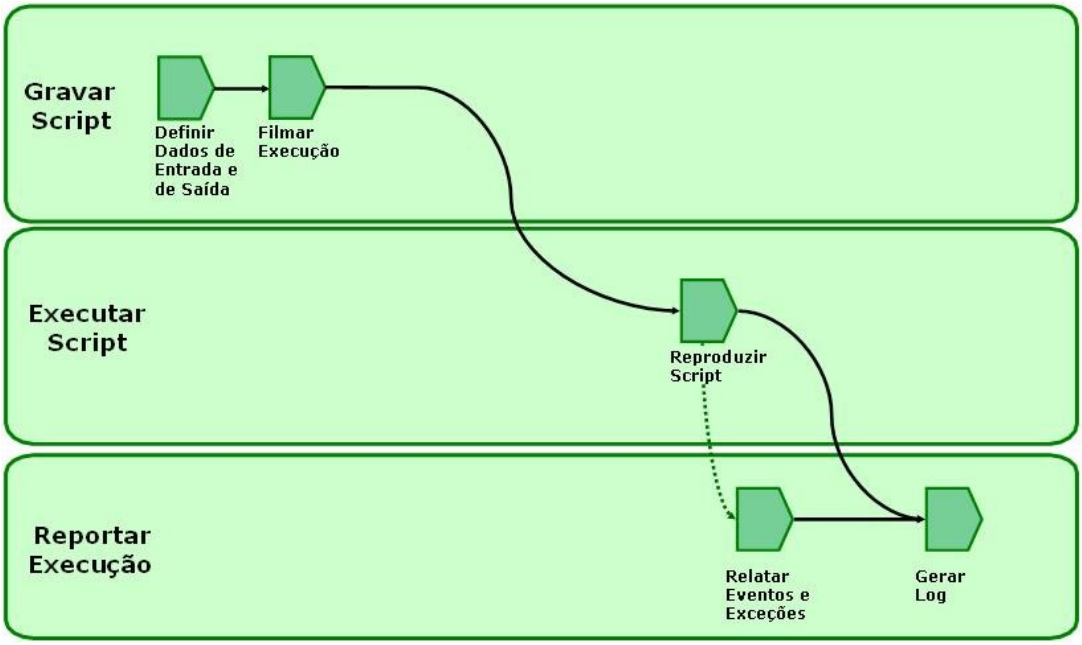

Figura 3 – Processo *Record & Playback*.

Este capítulo irá explicar como o Rational Functional Tester oferece suporte para execução de cada macro-tarefa e cada uma de suas tarefas internas:

3.2.1. Gravar Script:

Esta macro-tarefa tem como objetivo gravar a execução do teste manual salvando as informações sobre os dados utilizados, as navegações e funções utilizadas e onde devem ser avaliados os resultados esperados do teste. Esta tarefa é dividida neste processo em três subtarefas como a seguir:

3.2.1.1. Definir Dados de Entrada e de Saída:

A automação de testes funcionais utiliza um documento chamado projeto de testes, ele contém todo o passo a passo a ser seguido e os dados de entradas e saídas esperadas do teste funcional.

Desta forma, é preciso definir na ferramenta quais serão os dados de entrada e de saída para cada caso de teste do sistema a ser automatizado.

As entradas e saídas dos testes podem ser colocadas em uma estrutura já fornecida pelo Functional Tester chamada *datapool*.

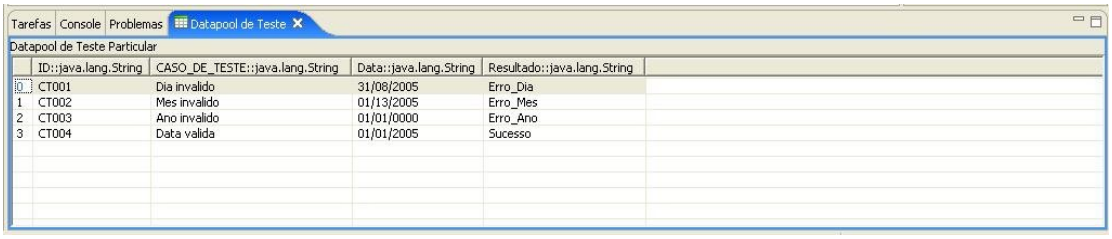

Figura 4 – Datapool do Rational Functional Tester.

Um *datapool* é uma série de dados de teste, uma coleção de registros de dados relacionados que fornece valores de dados às variáveis em um script de teste durante a sua execução. O uso de *datapools* pode fornecer uma separação explícita entre dados e funções [3].

3.2.1.2. Filmar Execução:

Após definido os dados de entrada e os resultados esperados na saída, é preciso fazer uma filmagem da utilização do software pelo usuário. Esta filmagem vai gerar um script contendo informações sobre os objetos e ações sobre estes. A filmagem é bastante importante para uma posterior reprodução deste script.

Quando se grava um teste de software em um script, o Functional Tester grava todas as ações do usuário utilizando a aplicação, tais como eventos de mouse e teclado, dados preenchidos em cada campo, etc.

Pode-se também introduzir pontos de verificação aos dados de teste ou às propriedades dos objetos em sua aplicação, onde estas propriedades ou dados serão avaliados a fim de se comprovar a corretude da aplicação. Durante a gravação, o ponto de verificação captura a informação do objeto e a armazena em um arquivo. Então durante a reprodução, o ponto de verificação captura a informação do objeto e a compara com o arquivo [3].

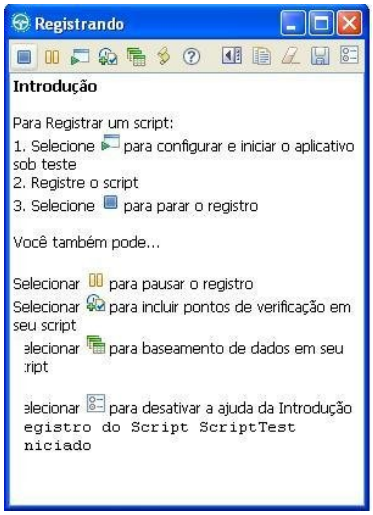

Figura 5 – Janela mostrada durante filmagem dos scripts.

3.2.2. Executar Script:

Quando se executa um script, o Functional Tester reproduz as ações gravadas anteriormente como a inicialização da aplicação, as ações que foram gravadas na aplicação e os pontos de verificação adicionados.

Para reproduzir um script no Functional Tester [3]:

1. Configure a aplicação sob teste ajustando o ambiente ou o navegador apropriado para executar a aplicação.
2. Execute o script de teste
3. O monitor do Playback inicializa e fornece informações enquanto o script executa.

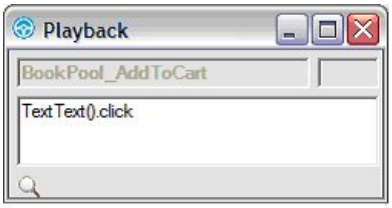

Figura 6 – Janela mostrada durante execução dos scripts.

Para realizar o reconhecimento de objetos da aplicação sob teste o RFT utiliza uma técnica chamada *ScriptAssure*. Cada objeto em um mapa do objeto de teste tem um conjunto de propriedades de reconhecimento que são estabelecidas durante a gravação [3].

Por exemplo, um botão com cinco propriedades de reconhecimento: nome, tipo, papel, classe e índice. Para encontrar um objeto na aplicação sob teste durante a reprodução, o Functional Tester compara o objeto na aplicação com as propriedades de reconhecimento no mapa de objeto de teste. Cada propriedade de um objeto de teste é associada a um peso. O Functional Tester usa o valor deste peso a fim de determinar a importância da propriedade para o reconhecimento do objeto.

3.2.3. Reportar Execução:

Durante a execução dos testes funcionais, o RFT monitora os resultados de cada ponto de verificação e as exceções que ocorrem no sistema. Após a execução dos scripts de teste, um log é gerado contendo todas estas informações.

3.2.3.1. Relatar Eventos de Exceção e Parar:

Durante a execução dos testes funcionais, o RFT fica monitorando os eventos que ocorrem no sistema.

O teste funcional registra automaticamente os seguintes eventos:

- Início do script
- Fim do script
- Chamadas de outros Scripts
- Chamadas ao método *startApplicaction*
- Início do *timer*
- Fim do *timer*
- Exceções
- Resultados dos Pontos de verificação (Falha ou Erro)

Caso ocorra alguma das quatro exceções abaixo, o sistema pára a execução e reporta o erro ocorrido no log da execução.

- *ObjectNotFoundException*
    - Ocorre quando algum objeto gravado não é encontrado na reprodução do script

- *AmbiguosRecognitionException*
    - Ocorre quando o Functional Tester não consegue distinguir entre os objetos da tela, qual foi usado durante a gravação (reconhecimento ambíguo)
- *CallScriptException*
    - Ocorre quando o Functional Tester não consegue inicializar algum script.
- *UnhandledException*
    - Ocorre quando uma exceção é lançada pelos scripts de teste e não é tratada.

3.2.3.2. Gerar Log:

Depois que a execução termina, é possível ver os resultados em um log. Os resultados incluem todos os eventos registrados na tarefa anterior, tais como falhas dos pontos de verificação, exceções no script, alertas sobre reconhecimento dos objetos e qualquer informação adicional da execução.

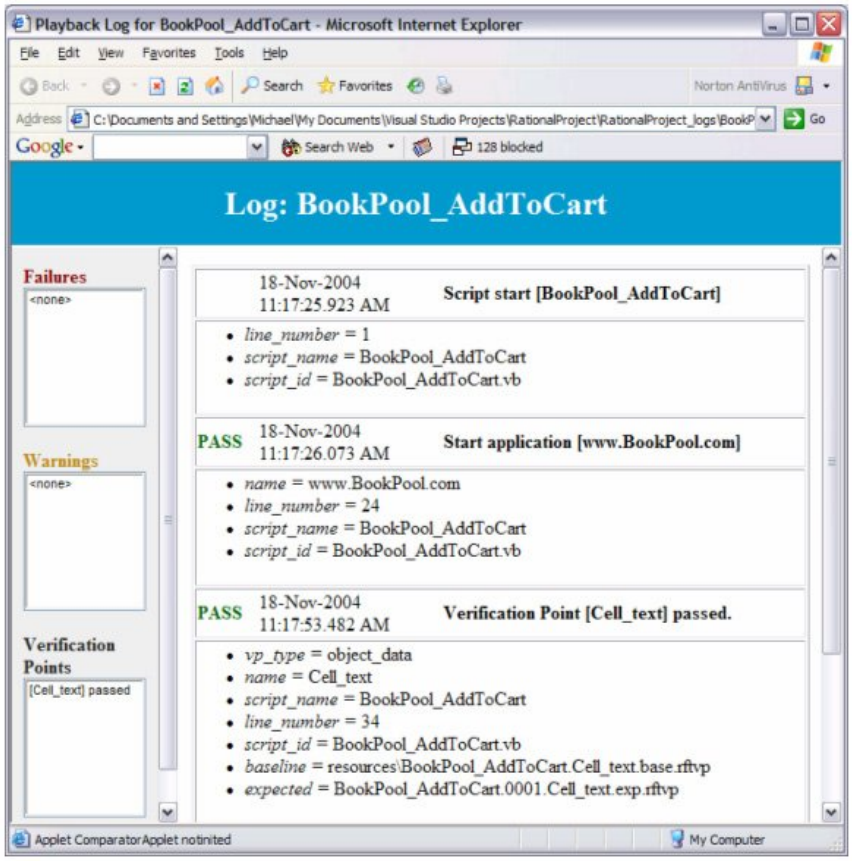

Figura 7 – Log gerado pela Execução.

3.3. Extensão do Rational Functional Tester

O Rational Functional Tester fornece recursos avançados que permitem aumentar a eficiência de algumas tarefas do processo apresentado sem mudar o processo em si. Esta seção irá explicar as modificações tanto na tarefa Filmar Execução quanto Reproduzir Script.

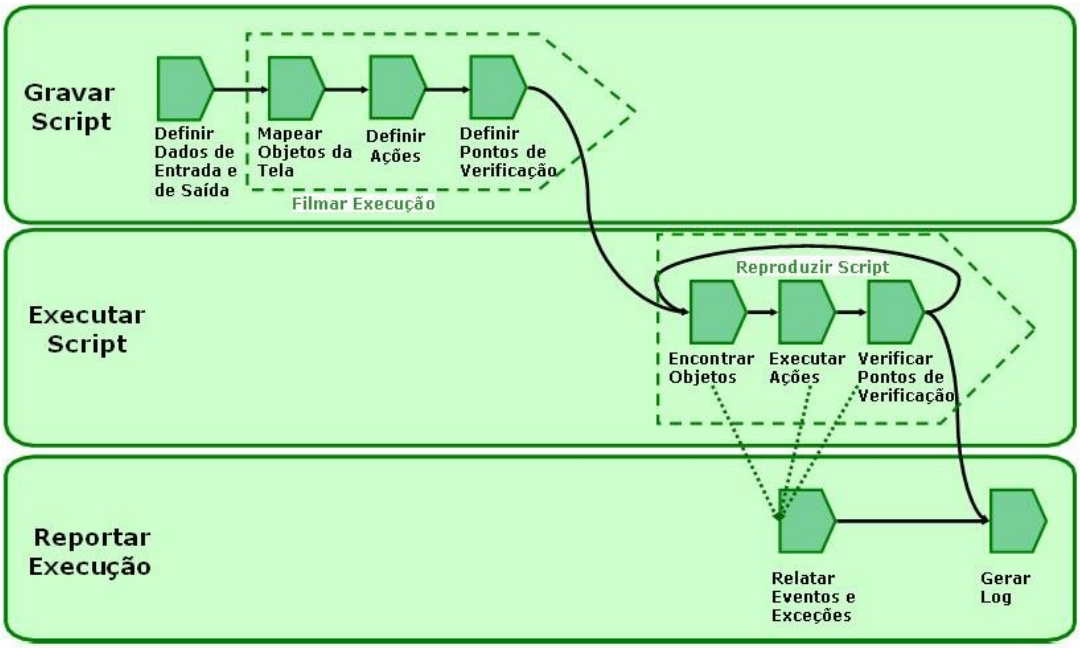

Figura 8 – Extensão do Processo *Record & Playback*

3.3.1. Extensão das tarefas de Gravar Script

Na macro-tarefa Gravar Script é possível refinar a tarefa Filmar Execução em três novas tarefas. Mapear Objetos, Definir Ações e Definir Pontos de Verificação. Estas tarefas são repetidas para cada objeto toda vez que eles são utilizados pelo usuário.

Inicialmente ele é mapeado juntamente com as ações efetuadas pelo usuário – clicar, digitar um texto, etc. – e por fim são definidos os pontos onde serão verificadas as propriedades ou os valores dos objetos para se atestar o funcionamento correto da aplicação sob teste.

A seguir será detalhado o funcionamento de cada uma das novas tarefas refinadas:

3.3.1.1. Mapeamento dos Objetos

Ao se gravar um script, o Functional Tester cria um mapa do objeto de teste – botões, telas ou programas – para a aplicação que está sendo testada. Este mapa funciona como uma lista estática que descreve todos os objetos de teste reconhecidos pelo Functional Tester na aplicação sob teste.

O mapa do objeto de teste contém propriedades de reconhecimento para cada objeto. Estas propriedades podem ser alteradas manualmente para se obter um melhor desempenho do reconhecimento destes objetos durante a execução dos scripts de teste, inclusive fazendo uso de expressões regulares para garantir o reconhecimento mesmo com pequenas mudanças nos valores das propriedades dos objetos.

3.3.1.2. Definir Ações:

Após se obter o mapeamento de um objeto, é preciso definir qual a ação que será realizada. Cada objeto mapeado pelo Functional Tester cria um objeto em Java que possui métodos que representam todas as ações habilitadas para aquele tipo de objeto. Exemplos de ações: clicar em determinado ponto do objeto, inserir texto, ler o texto digitado no objeto, marcar como selecionado, desmarcar, etc.

3.3.1.3. Definir Pontos de Verificação:

Os pontos de verificação verificam se alguma ação ocorreu, ou verificam o estado de um objeto. Ao se criar um ponto de verificação, o sistema captura a informação sobre o objeto na aplicação para estabelecê-lo como referência para a comparação durante a reprodução do script [3].

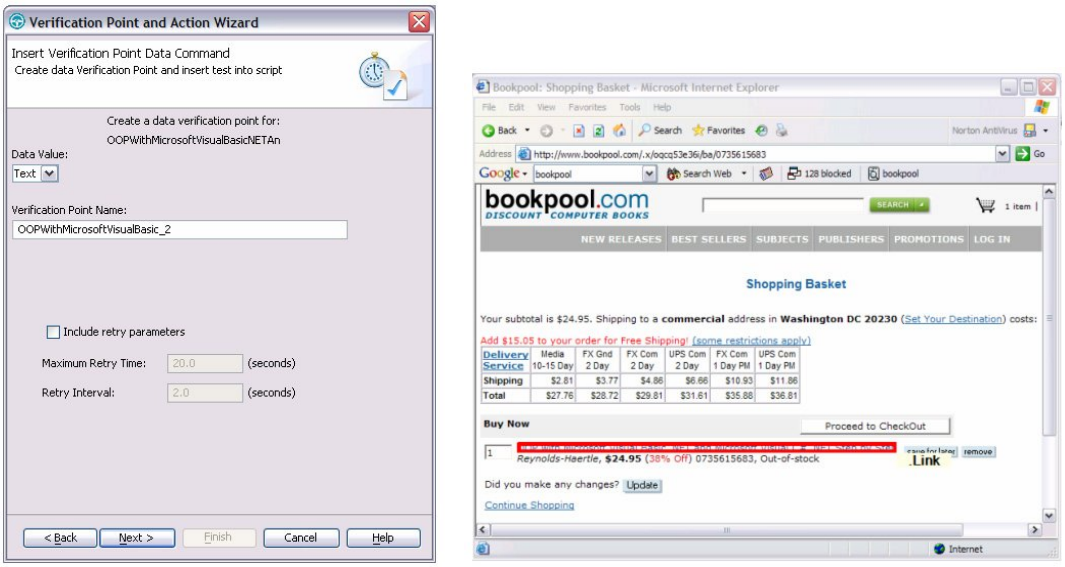

Figura 9 – Janelas mostradas durante a adição de um ponto de verificação.

3.3.2. Extensões das tarefas de Reproduzir Script

A tarefa Reproduzir Script dentro da macro-tarefa Executar Script pode ser dividida em três tarefas da mesma forma que a tarefa Filmar Script foi dividida.

3.3.2.1. Encontrar Objetos

Para que o Functional Tester reconheça um objeto na aplicação sob teste, as propriedades do objeto devem combinar – *matching* – com as propriedades gravadas no mapa do objeto do teste. Por padrão, o Functional Tester pode encontrar o objeto se uma ou duas propriedades não combinarem, porém o RFT reportará no log de execução um alerta sobre o reconhecimento fraco. Se mais de três propriedades não combinarem, o Functional Tester não vai encontrar o objeto na aplicação.

Com a manipulação do mapeamento dos objetos, é possível executar scripts com sucesso mesmo quando a aplicação sob teste for atualizada. Se os objetos na aplicação mudarem, é possível usar a funcionalidade do *ScriptAssure* para controlar a sensibilidade da comparação entre objetos.

3.3.2.2. Executar Ações

Depois de identificado o objeto na tela, o Functional Tester irá tentar reproduzir a ação filmada durante a gravação dos scripts. Se não for possível realizar

a ação, o RFT lançará uma exceção que será adicionada ao log da execução dos scripts.

3.3.2.3. Verificar Pontos de Verificação

Os pontos de verificação definidos na gravação dos scripts de teste são executados nesta tarefa. Caso o objeto a ser avaliado não seja encontrado uma exceção de *ObjectNotFoundException* pode ser lançada. Caso contrário, o objeto será avaliado e o resultado reportado no log da execução dos scripts.

3.4. Considerações Finais

Este capítulo descreveu as principais funcionalidades do Rational Funcional Tester seguindo um processo *Record & Playback*. A ferramenta possui boas funcionalidades para o processo, porém este processo limita muito o desenvolvimento de scripts de teste com fácil leitura pelo desenvolvedor e uma boa manutenção.

Os scripts possuem muita repetição de código e nenhum reuso. A geração do código feita automaticamente pela gravação das ações do usuário pela ferramenta, não deve ser utilizada irrestritamente.

Outros fatores limitantes da ferramenta são as formas de entrada e saídas de dados feitas, respectivamente, pelo datapool e pelo log da execução. Estas duas funcionalidades limitam bastante a automação de testes que necessitam uma facilidade de atualização das entradas dos scripts e uma rapidez na coleta dos dados da execução.

Como solução destes problemas, o próximo capítulo irá propor um *framework* para automação de testes que faz modificações tanto no processo quanto nas tarefas propostas neste capítulo a fim de produzir um melhor resultado na automação de testes funcionais.

4. *Framework* para automação de testes funcionais

O *framework* proposto utiliza uma técnica de construção e organização dos scripts de teste que faz mudanças no processo e nas tarefas do Rational Functional Tester. A utilização do *framework* provê mais reuso e independência entre os scripts e melhora a robustez da execução dos casos de testes, além de melhorar de uma maneira geral a manutenibilidade dos scripts de teste. Este capítulo falará das melhorias feitas no processo de automação de testes e na arquitetura do sistema para implementar tais melhorias.

4.1. Melhorias no processo

As melhorias no processo *Record & Playback* foi definido tendo como base o trabalho de Zambelich em [17], que será explicado a seguir:

4.1.1. Método de decomposição funcional

A técnica de criação de scripts por decomposição funcional visa reduzir todos os casos do teste a suas tarefas mais fundamentais. Ele consiste em escrever as funções relativas ao usuário, os scripts das regras de negócio do caso de teste e os scripts de utilidades gerais que executam estas tarefas independentemente uma das outras. Estas áreas fundamentais incluem:

1. Navegação
2. Funções Específicas (do Negócio)
3. Verificação
4. Navegação de Retorno

Para realizar tal objetivo é necessário separar os dados das funções. Isto permite que um script automatizado de teste seja escrito para executar uma função

do negócio, usando arquivos para fornecer os dados de entrada e dos resultados esperados.

O motor da execução é o Driver Script que contém uma série de chamadas a um ou mais scripts de teste que vão realmente executar o teste automaticamente. Os scripts de caso de teste contêm a lógica do caso do teste, mas quem vai realmente testar a aplicação são os scripts da regra do negócio que contêm todas as funções do sistema mapeadas. Os scripts de utilidades gerais são chamados de acordo com a necessidade dos scripts principais.

Cada script tem o seguinte objetivo:

Driver Scripts:

Executa a inicialização, se necessário, da aplicação e chama os scripts de caso do teste na ordem desejada.

Scripts de caso de teste:

Executa a lógica do caso de teste da aplicação, usando scripts da função do negócio.

Scripts da regra de negócio:

Executa funções específicas do negócio dentro da aplicação.

Scripts de utilidade geral:

Executa as tarefas específicas da aplicação requeridas por dois ou mais scripts de teste.

Funções relativas ao Usuário:

Funções gerais, específicas da aplicação, e de acesso a telas. Estas funções podem ser chamadas por qualquer um dos scripts acima citados.

Os próximos passos representam um caso de teste de um pagamento.

1. Acesse a tela de pagamento através do Menu Principal
2. Faça o pagamento
3. Verifique se o pagamento atualizou o saldo da conta
4. Retorne ao menu principal
5. Acesse a tela de extrato através do Menu Principal
6. Verifique a atualização do saldo da conta
7. Acesse a tela do histórico das operações através da tela de extrato

8. Verifique a atualização do histórico das operações
9. Retorne ao menu principal

Um script de “regra de negócio” e um script de “utilidade geral” podem ser escritos como a seguir:

Pagamento:

- Inicie no menu principal
- Chame o método de navegação para acessar a tela de pagamento
- Leia o arquivo de dados contendo os dados específicos para o teste e insira esses dados
- Pressione o botão ou execute a função para efetuar o pagamento
- Leia o arquivo de dados contendo os dados de saída esperados
- Compare esses dados com os dados mostrados na tela (saídas atuais)
- Escreva qualquer discrepância em um relatório de erros
- Pressione o botão ou o link de retorno ao menu principal, ou se necessário, execute a função de navegação para fazê-lo.

Verificar Conta (Verificar Extrato e Histórico das operações):

- Inicie no menu principal
- Chame o método de navegação para acessar a tela de extrato
- Leia o arquivo de dados contendo os dados de saída esperados
- Compare esses dados com os dados mostrados na tela (saídas atuais)
- Escreva qualquer discrepância em um relatório de erros
- Pressione o botão ou link para acessar o histórico das operações
- Leia o arquivo de dados contendo os dados de saída esperados
- Compare esses dados com os dados mostrados na tela (saídas atuais)
- Escreva qualquer discrepância em um relatório de erros
- Pressione o botão ou o link de retorno ao menu principal ou, se necessário, chame a função de navegação para fazê-lo.

Os scripts da regra de negócio e de utilidade geral chamam as funções que simulam o usuário para efetuar a navegação. O script do caso de teste deve chamar esses dois scripts toda vez que precisar executar métodos de regras de negócio ou de utilidade geral.

O Driver Script, por sua vez, deve chamar o script do caso de teste pelo número de vezes que for necessário para executar todos os casos de teste deste tipo. Para cada tipo, a única coisa que muda são os dados contidos nos arquivos que são lidos e processados pelos scripts da regra de negócio e das sub-rotinas. O fluxo do teste é praticamente o mesmo, só mudando os dados de entrada e os de saída, podendo ser vários testes de sucesso ou de erro.

Usando este método, se for necessário processar cinqüenta tipos diferentes de pagamentos para verificar todas as possibilidades, será necessário apenas quatro scripts para modelar e executar todos os cinqüenta casos. Os scripts são:

1. O Driver Script
2. O script do caso de teste
    a. “Efetuar um Pagamento e Verificar os Resultados”
3. O script das regras de negócio de “Pagamento”
4. O script de utilidade geral
    a. “Verificar Extrato da Conta e Histórico das Operações”

Se fosse usado o método *Record & Playback,* iriam ser produzidos cinqüenta scripts diferentes, cada um com diferentes dados de entrada e saída no próprio corpo dos scripts, o que iria dificultar a manutenibilidade desses scripts.

Este método, porém, requer apenas que se adicionem os arquivos com os dados de entrada e saída de cada teste. Estes dados podem ser facilmente mantidos e atualizados devido a sua centralização. A atualização pode ser feita por pessoas que não detém conhecimento em ferramentas de automação, scripts ou programação. Isto significa que testadores sem conhecimento técnico podem exercer estas funções enquanto um testador mais técnico pode criar e manter scripts automatizados.

É importante salientar também que os scripts de utilidades gerais, os quais verificam o extrato da conta e o histórico das operações, podem também ser

utilizados por outros scripts de caso de teste e de funções da regra de negócio. Um estorno de pagamento, por exemplo. Se fosse necessário testar cinqüenta estornos de pagamentos seria necessário desenvolver apenas três scripts adicionais:

1. O Driver Script
2. O script do caso de teste
    a. "Estornar Pagamentos e Verificar Resultados"
3. O script da regra de negócio do "Estorno de Pagamento"

Considerando que já existem os quatro scripts anteriores, é possível rapidamente desenvolver estes três a partir dos originais. É possível utilizar o script de utilidades gerais, "Verificar Extrato da Conta e Histórico das Operações", sem nenhuma modificação.

Se for necessário utilizar diferentes contas, será preciso apenas atualizar os arquivos de dados ao invés de atualizar o script todo. Por este e outros motivos, este método se torna mais lucrativo do que o método *Record & Playback*.

4.1.2. Implementação da Técnica

Para implementação da técnica foi usado como inicio o processo *Record & Playback* mostrado anteriormente, modificando conforme a necessidade.

Em relação ao processo, as principais mudanças foram na macro-tarefa Executar Script. A tarefa "Executar Navegação, se necessário" foi adicionada para permitir a independência da execução entre casos de testes. Esta tarefa pode ser chamada depois do relato de alguma exceção ou pela própria execução dos casos de teste.

Outra importante mudança no processo é a redefinição da tarefa "Relatar Eventos e Exceção" que não pára mais a execução.

No inicio do processo, os dados de entrada e saída são lidos agora diretamente das planilhas de projeto de caso de teste. Além disto, a utilização de planilhas também ocorre na reportagem da execução na última tarefa do processo: "Gerar Planilhas".

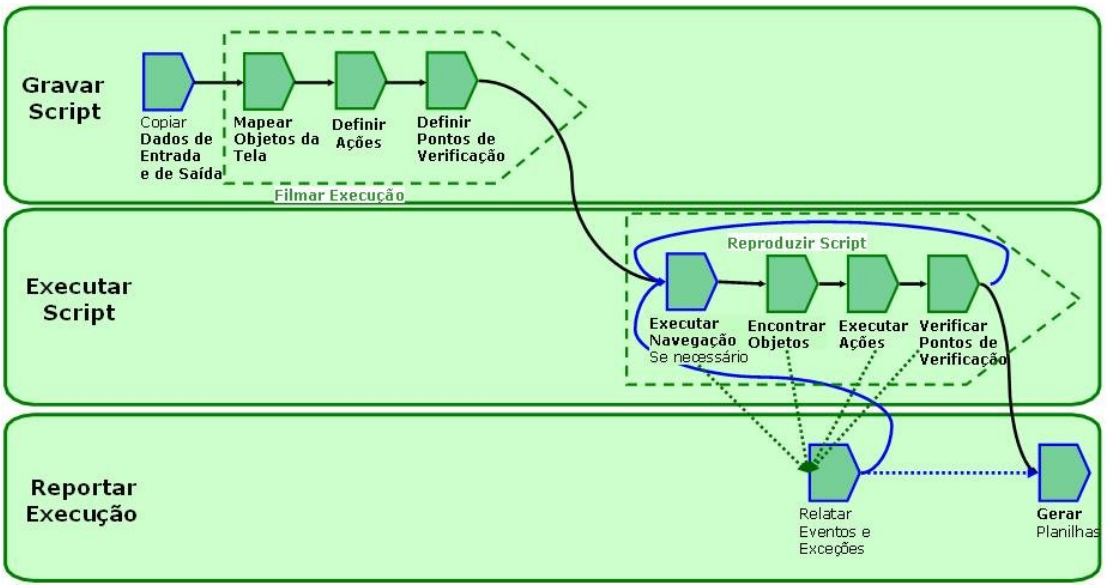

Figura 10 – Processo de Decomposição Funcional

Uma importante mudança no processo foi que a filmagem dos scripts não ocorre mais como no processo Record & Playback quando todos os scripts de teste eram gerados a partir de gravações. Seguindo o novo processo, as filmagens ocorrem apenas para gerar os scripts auxiliares – funções da regra de negócio e funções de utilidades gerais – que serão utilizados pelos scripts de caso de teste, como será visto na próxima seção.

4.2. Arquitetura do Sistema

O *framework* proposto neste trabalho é composto por quatro scripts principais e dois subsistemas de acordo com a figura a seguir:

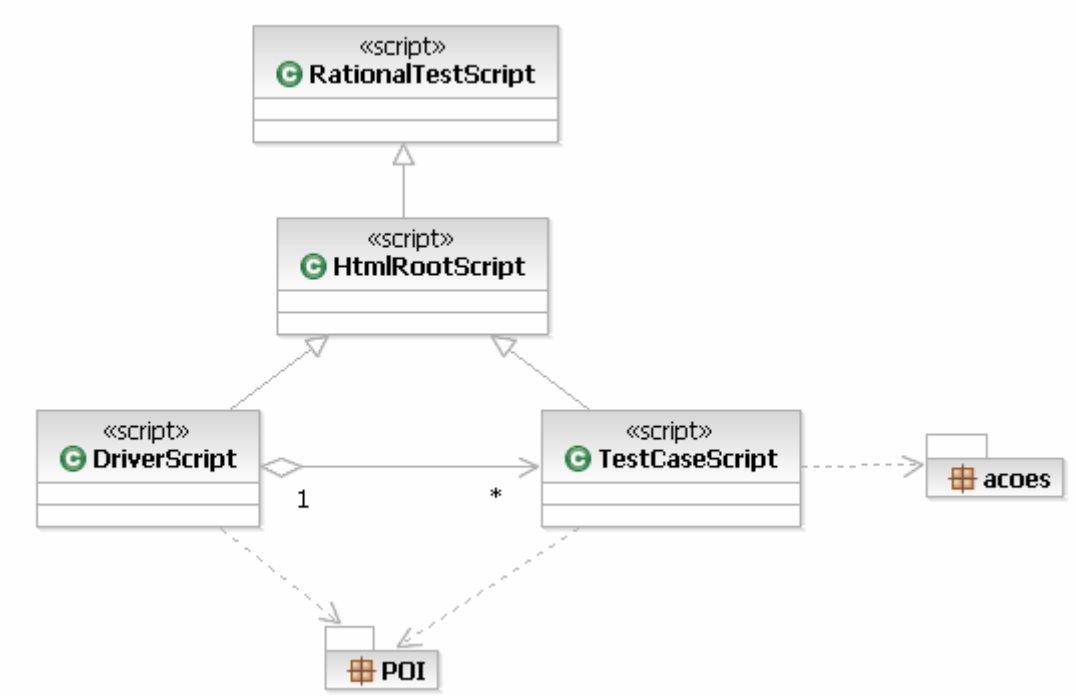

Figura 11 – Arquitetura do Sistema

O primeiro script é o RationalTestScript, que fornece todas as funcionalidades básicas para a utilização dos scripts de teste no Functional Tester. Este script disponibiliza para utilização e extensão funcionalidades básicas de leitura e escrita em *datapools*, identificação e manipulação de objetos na tela, além de reportagem de eventos e tratamento de exceções durante a execução.

Este script é fornecido pelo Functional Tester e apresenta as funcionalidades ainda com pouco detalhamento. As funcionalidades precisam ser genéricas o bastante para se adaptar a qualquer tipo de aplicação sob teste, mas para o *framework* proposto, foi feito uma especialização desta classe para se obter resultados mais precisos nas diversas funcionalidades oferecidas pela ferramenta.

Um script raiz precisa ser criado para conter funcionalidades específicas do processo de testes adotado e da aplicação sob teste. Como no estudo de caso a aplicação a ser testada é uma aplicação *web*, criou-se o HtmlRootScript. Este script serve de base para todos os outros scripts de caso de teste, pois possui funcionalidades modificadas para reconhecimento e manipulação de objetos, tratamento de eventos e exceções, etc.

Depois de criado o script raiz, o DriverScript foi criado para guiar a execução dos scripts de caso de testes. O DriverScript é responsável pela chamada a todos os scripts de teste de acordo com a suíte de execução escolhida pelo testador.

O TestCaseScript representa um script de caso de testes. Ele contém os códigos de fluxo da execução dos testes automáticos e toda vez que é necessário, ele utiliza os scripts do pacote "acoes" que são os scripts que realmente utilizam o software sob teste. Abaixo segue um exemplo dos scripts disponíveis neste pacote:

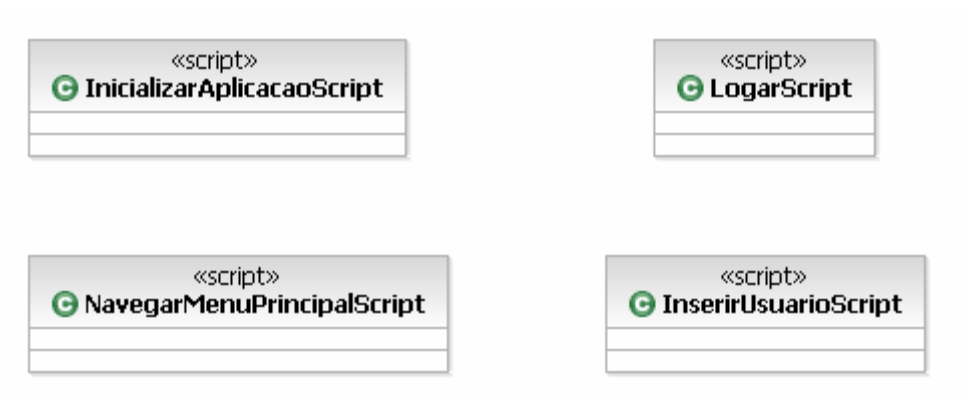

Figura 12 – Arquitetura do pacote acoes

Dentro do pacote "acoes" estão os scripts de funções da regra de negócio e de propósito geral que serão usados por scripts de caso de teste. Esta modularização traz benefícios para a manutenibilidade dos scripts de teste e para a robustez da execução dos scripts.

A manutenibilidade é beneficiada pela concentração em um único local das funções atreladas à regra de negócio da aplicação sob teste. Operações como inicialização da aplicação, navegação até o menu principal, entre outras, ficam concentradas em um único local, deixando os scripts de teste mais auto-contidos.

Já a execução é beneficiada pela independência entre os scripts de teste. Se um script de navegação falhar, o script de caso de teste ou até mesmo o DriverScript será capaz de reportar o erro e chamar as funções de inicializar aplicação e de navegação até o menu principal para dar continuidade à execução dos próximos scripts.

Para fazer a reportagem dos eventos e exceções da execução, foi utilizado o componente Jakarta POI [5], uma biblioteca Java para acessar arquivos no formato Microsoft OLE 2, como arquivos Word e Excel. Este componente possibilitou o acesso aos dados de entrada diretamente das planilhas de projeto dos casos de teste. A classe GerarDatapool associa um datapool privado a um script de caso de teste e o preenche com os dados da planilha de projeto de teste. Desta maneira, para alterar algum dado de entrada basta alterar a planilha de projeto de teste e chamar as funções da classe CopiarDados para atualizar o *datapool.*

O pacote POI também é utilizado para fazer a reportagem da execução de teste. Após a execução de cada caso de teste, o HtmlRootScript utiliza a classe ReportarExecucao para reportar todos os eventos e exceções ocorridas na planilha de execução de teste em planilhas.

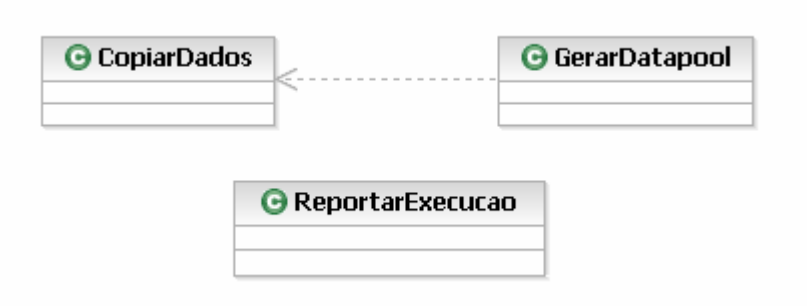

Figura 13 – Arquitetura do pacote POI

4.3. Considerações Finais

Este capítulo descreveu a técnica de criação de scripts por decomposição funcional que visa a criação de scripts de negócio, de utilidade geral, scripts de casos de teste e um script para comandar a execução. Esta arquitetura permite que se construam scripts mais enxutos, facilitando a manutenção e provendo uma independência entre esses scripts durante a execução, fundamental para uma execução robusta.

No próximo capítulo serão apresentados os resultados da utilização deste *framework* em um projeto de desenvolvimento de software em uma fábrica de software.

5. Resultados Obtidos

Para testar o *framework* proposto no capítulo 4, foi realizado um experimento em uma suíte de teste selecionada. O objetivo do experimento é comparar a efetividade da automação realizada utilizando o *framework* com o processo de execução manual utilizado no passado. Neste capítulo serão descritos o projeto alvo do experimento, o experimento realizado e avaliação dos resultados encontrados.

5.1. Contexto

O alvo deste experimento é um projeto de um sistema *web* de uma fábrica de software. Este projeto foi desenvolvido por vinte oito desenvolvedores e analistas, quatro projetistas de testes e três automatizadores de testes durante um ano.

Atualmente, praticamente 75% dos testes funcionais são realizados de forma manual. O esforço empenhado pelos testadores em realização de testes de regressão representa cerca de 30% da força de trabalho da organização. A utilização de testes automáticos na execução de testes é uma das propostas da organização para redução deste esforço.

5.1.1. Processo de Testes

O processo utilizado para testes funcionais é feito em três etapas:

1) Definir Plano de Testes
2) Definir Projetos de Testes
3) Executar Testes

Na primeira etapa de Definição do Plano de Testes são definidas questões como:

- Alocação de recursos
- Definição dos ciclos de execução de testes
- Estratégias utilizadas para os testes de software
- Escopo dos Testes
- Cronograma dos Testes

Durante a segunda etapa, os projetos de testes são definidos e documentados os casos de testes baseados nos documentos de requisitos do projeto.

Na última etapa, de execução dos testes, os projetos de casos de testes são utilizados para a execução manual dos testes projetados.

Este tipo de abordagem pode levar a um excesso de tempo gasto na execução dos testes, deixando de re-executar testes passados quando novas funcionalidades são adicionadas ao projeto.

Testes funcionais que serão executados diversas vezes podem dar maior e melhor resultado se forem automatizados, conforme veremos no relato do experimento realizado neste trabalho.

5.1.2. Escopo do Experimento

O experimento usou como amostra uma suíte de uma das aplicações testadas pela organização. A aplicação em questão, denominada A1 possui 64 suítes de teste, formando um conjunto de 7005 casos de teste. Os ciclos de teste são compostos, normalmente, por aproximadamente 2200 casos de teste.

O último ciclo de regressão realizado pela organização em setembro de 2006 possuía 2219 casos de teste, demandando um esforço considerável para sua execução.

A automação de parte destes testes pode trazer um grande retorno e economia de esforço para a organização. Para a realização do experimento foi necessário selecionar uma suíte da aplicação A1. A suíte de testes que foi alvo de automação, denominadas aqui de S1, possui 791 casos de teste dos quais 732 foram automatizados em três meses.

A automação de testes modificou o processo utilizado pela empresa colocando mais um passo entre as tarefas de Definir Projetos de Testes e Executar Testes. Agora, existe uma tarefa para Automatizar os Testes, que consiste no desenvolvimento de scripts de teste que irão executar automaticamente na tarefa de execução dos testes.

5.2. Análise dos Resultados

Para avaliar resultados obtidos através da aplicação do *framework* proposto no processo de testes da fábrica de software será calculado o retorno do investimento,

utilizando como guia – mas com algumas mudanças que serão explicadas mais adiante – o artigo de VARGHESE sobre retorno de investimentos de projetos de automação de testes de software [16].

Para realizar uma análise comparativa entre a abordagem manual e automatizada alguns dados e premissas devem ser considerados:

- Após a automação de testes algum esforço deve ser considerado para sua manutenção corretiva e evolutiva. Foi levado em conta um tempo equivalente a 2% do esforço de automação para manutenção dos casos de teste automatizados antes de cada ciclo de execução quinzenal.
- Esta análise considera que o custo homem/hora de um testador equivale ao custo homem/hora de um programador dedicado à automação de testes que é de três mil reais mensais, incluso salário e encargos, ou R$18,75 por hora.
- O hardware usado tanto para automatizar os testes, quanto para executar os testes manuais e automáticos são os mesmo, um PC comum de dois mil reais.
- O cálculo de retorno de investimento considera igual a necessidade de espaço físico para executar testes manuais ou automáticos, o que pode não ser verdade em alguns projetos.

A seguir serão levantados todos os custos e melhorias da área de testes devido à automação para se fazer o cálculo do tempo de retorno do investimento em automação feito na fábrica de software.

5.2.1. Custos da Automação

A automação de testes envolve custos fixos e variáveis. Os componentes dos custos fixos, os que ocorrem uma única vez para cada esforço de automação, estão listados na tabela 1. Na maioria das vezes, este componente terá uma participação considerável no custo total do esforço de automação.

Os custos fixos são compostos por:

- Custo da ferramenta: uma licença do Rational Functional Tester mais doze meses de suporte incluso.

- Custo para construção dos scripts: utilização de um automatizador por um período de três meses, salários e encargos.
- Esforço tanto para testar e validar, quanto para documentar os scripts de teste: em média 10% do esforço para sua construção.
- Custo da infra-estrutura: um computador pessoal para ser usado pelo automatizador de teste.

Custos Fixos	Valores
Custo da ferramenta	R$ 21.150,50
Esforço de script de teste	R$ 9.000,00
Esforço de testar e validar os scripts	R$ 900,00
Custo da infra-estrutura de automação	R$ 2.000,00
Esforço de documentação das suítes de teste	R$ 900,00
Total	R$33.950,50

Tabela 1 – Custos Fixos da Automação de Testes

A tabela 2 lista os componentes do custo variável da automação. Estes custos aconteceram para todos os ciclos de testes que executarem os testes automatizados durante o desenvolvimento do software em questão. Estes custos são:

- Custo da manutenção dos scripts: em média 2% do custo de sua criação.
- Custo da infra-estrutura: o custo de manter o computador da automação, 10% do preço inicial, onde irão rodar todos os testes automatizados.
- Custo da execução: o custo do automatizador durante duas horas para fazer a coleta dos dados do relatório da execução dos testes automatizados.

Custos Variáveis	Valores
Manutenção dos scripts e de sua documentação	R$ 180,00
Manutenção da infra-estrutura da automação	R$ 200,00
Custo da execução	R$ 37,50
Total	R$417,50

Tabela 2 – Custos Variáveis da Automação de Testes

5.2.2. Custos dos Testes Manuais

Como tanto a entrada da execução dos testes manuais quanto da automação dos testes são os projetos dos casos de testes, os únicos custos associados aos testes são os custo da própria execução manual e de sua infra-estrutura. Estes custos são:

- Custo da execução: Segundo histórico da fábrica de software onde foi feito o experimento, o tempo médio de uma execução de teste manual é de 2 minutos e 18 segundos, que multiplicado pela quantidade de casos de testes, 791, dá um total de 30,32 horas com um valor de R$ 18,75 por hora.
- Custo da infra-estrutura: o custo da manutenção de cinco computadores para execução dos testes manuais.

Custos Variáveis	Valores
Manutenção da infra-estrutura da execução manual	R$1.000,00
Custo da execução	R$ 568,50
Total	R$1.568,50

Tabela 3 – Custos Variáveis dos Testes Manuais

5.2.3. Melhorias na execução dos testes

Para calculo dos benefícios da automação de testes é necessário calcular a melhoria no custo de execução dos testes automatizados em comparação com os testes manuais.

O custo inicial para o desenvolvimento dos scripts de teste faz parte do custo fixo dos testes automatizados, pois este valor será diluído a cada novo ciclo de testes executado.

A diminuição do custo da execução de testes ocorrerá principalmente pela redução do tempo da execução dos testes.

5.2.3.1. Melhorias no tempo de execução dos testes

Como dito anteriormente, o tempo total de execução da suíte é de 30,32 horas ou 109.158 segundos.

Já o tempo médio da execução de um teste automatizado foi obtido a partir da execução de 732 casos de testes que levaram ao todo quatro horas e dez minutos para

executar. Tirando uma média do tempo da execução de todos os casos de testes, chegamos ao valor de 20,49 segundos para execução de cada caso de testes automatizados. Mas o valor para efeito de comparação deve ser o valor da execução da suíte inteira. Devemos levar em consideração também os cinqüenta e nove casos de testes que não puderam ser automatizados.

Desta forma, o tempo total da execução dos testes após automação são quatro horas e dez segundos da execução dos testes automatizados, ou 14.410 segundos, somado com 8.142 segundos que dá um total de 23.992 segundos.

Calculando a melhoria entre a execução manual e automática, chegamos ao valor de 78,02% de melhoria no tempo da execução.

5.2.3.2. Melhorias no custo de execução dos testes

Para calcular a melhoria nos custos da execução dos testes, podemos simplesmente diminuir o custo da execução manual pela execução automatizada e dividir pelo custo da execução manual.

Tendo um custo de execução manual equivalente à R$1.568,50 e o custo da execução automatizada de R$417,50, temos uma melhora de 73,38% de melhoria no custo da execução.

Este benefício será utilizado para calcular o retorno do investimento da automação de testes de software.

5.2.4. Retorno do Investimento

A automação de testes é uma melhoria do processo de teste. Isto não deve ser visto apenas como uma atividade de engenharia que é encaixada como parte da atividade de teste. A eficácia e a eficiência deste processo devem ser medidas e validadas baseadas em objetivos organizacionais.

Para se fazer tal validação será calculado o retorno do investimento para o projeto da automação com uma fórmula um pouco diferente da apresentada no artigo de VARGHESE [16].

A fórmula descrita no artigo não leva em conta algumas variáveis importantes no cálculo do ROI de um projeto de automação de testes. Para calcular o benefício de automação o artigo utiliza a variável melhoria do tempo da execução. Porém, a

variável correta seria a melhoria no custo da execução, porque se uma execução automática leva 10% do tempo de uma execução manual, mas utiliza um servidor muito caro e sofisticado que custa um milhão de dólares, esta variável deveria entrar na conta do retorno do investimento. Outro fator que reforça a mudança da variável é que o tempo da execução já influi no seu custo, por isso a variável custo é muito mais representativa do que o tempo.

Para se fazer tal validação será calculado o retorno do investimento para o projeto da automação a partir da seguinte equação:

N = F / (M * S) , onde:

Variável	Sigla
Nº. de ciclos para o ROI	N
Custo fixo da automação	F
Custo dos testes manuais por ciclo	M
% da redução de custo da execução dos testes	S

Tabela 4 – Variáveis do ROI da automação de testes

Utilizando a formula acima para calcular o ROI do projeto de automação, somente com a S1 automatizada, temos os seguintes valores:

Nº. de ciclos = C. fixo / (C. dos testes manuais * % da redução de custo)

Nº. de ciclos = 33.950,50 / (1.568,50* 73,38%)

Nº. de ciclos = 33.950,50 / 1.150,96

Nº. de ciclos = 29,49 ciclos

O gráfico abaixo mostra os custos da execução manual em comparação com o custo total da automação. Quando o gráfico da execução manual se encontra com o da automação, ocorre o *break-even*, ou o retorno do investimento na automação. Neste momento todo o dinheiro investido em automação foi recuperado devido à economia feita com os testes manuais, o que segundo os cálculos acima ocorre com 29,49 ciclos de execução de testes.

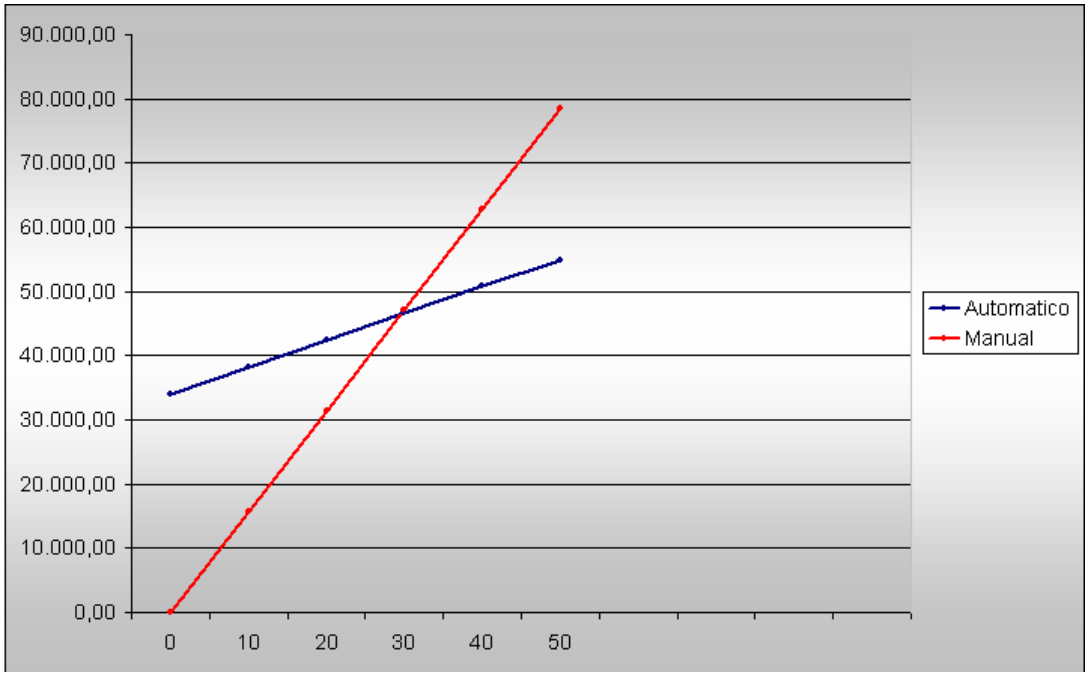

Figura 14 – Comparação dos custos de execução automática e manual por ciclo

5.3. Considerações Finais

Este capítulo apresentou o relato do experimento realizado para comprovar a eficiência do *framework* proposto na automação de casos de teste.

O experimento comparou o investimento realizado e o tempo necessário para retorno deste investimento ao selecionar casos de teste utilizando o método proposto com os dados relativos à execução de testes sem a aplicação da automação dos testes.

O *framework* mostrou-se bastante eficaz ao reduzir significativamente a necessidade de esforço da execução dos testes, com uma melhoria do tempo de execução de 78,02% e de custo da execução de 73,38%.

Do ponto de vista do retorno do investimento em automação, é possível visualizar claramente que o *framework* utilizando a ferramenta proposta foi um investimento um pouco alto, com um retorno de investimento em aproximadamente 29 ciclos de testes. Levando em conta o histórico da fábrica de software analisada, em um ano de desenvolvimento foram executados vinte e quatro ciclos de testes, o que daria um retorno de investimento em aproximadamente um ano e três meses.

Apesar de ter melhorias tão expressivas nos números da execução dos testes é possível perceber que as variáveis para se decidir entre automatizar ou não um projeto de testes devem ser bem mais elaboradas do que somente a redução do tempo e do custo da execução dos testes.

6. Conclusões e Trabalhos Futuros

Este trabalho possibilitou, baseado na experiência do desenvolvimento da automação dos testes funcionais em uma fábrica de software, confirmar a importância da utilização de *frameworks* para auxiliar na automação de testes de software. O uso da técnica de decomposição funcional também auxiliou na produção de um conjunto de componentes enxutos e que contemplavam os requisitos dos projetos de teste.

O *framework* se mostrou eficiente com a construção de scripts que utilizam melhor o reuso de código e possuem uma maior independência de execução, o que permitiu executar todos os testes ininterruptamente sem que precisasse alguém ficar observando a execução dos testes.

Este trabalho permitiu também fazer uma análise detalhada da ferramenta Rational Functional Tester e dos recursos oferecidos por ela para o desenvolvimento de scripts para automação de testes. Mesmo com todas as deficiências, o Rational Functional Tester mostrou-se ser uma plataforma com bastante potencial para o desenvolvimento de scripts de teste, com suporte a vários recursos, e a possibilidade de extensão de diversas funcionalidades bastante interessantes.

6.1. Dificuldades Encontradas

As principais dificuldades encontradas no desenvolvimento deste trabalho foram com relação a grande necessidade de conhecimento da linguagem de script utilizado pela ferramenta para se obter o máximo de benefício dos recursos da linguagem e da ferramenta e a fragilidade do reconhecimento de objetos na tela por todas as ferramentas utilizadas neste trabalho.

Outro aspecto é o fato de os engenheiros de teste além de manter o plano de teste com os todos os dados específicos de testes, devem replicá-los nos vários arquivos de dados que serão utilizados pelos scripts de teste, isto dificultou um pouco a manutenção dos scripts de teste.

6.2. Trabalhos Futuros

Para dar continuidade a este trabalho, podemos identificar as seguintes utilizações do *framework* proposto:

- Utilizar o *framework* em outros projetos de testes de software para se analisar o impacto tanto no tempo e custo da execução quanto no tempo de retorno do investimento.
- Melhorar a identificação dos objetos através de funções específicas para o domínio da aplicação sob teste. A identificação de objetos de interface gráfica continua sendo uma tarefa muito difícil que diminui consideravelmente a viabilidade e rentabilidade da automação de testes.
- A possibilidade de se fazer testes na fachada. Os testes funcionais dos métodos da fachada são testes muito fáceis de automatizar e aumentam consideravelmente a cobertura dos testes funcionais em sistemas que utilizam tal padrão de projeto.

Referências

[1] FEWSTER, M., GRAHAM D., Software Test Automation, Addison-Wesley, 1999.

[2] HENDRICKSON, E., Build or Buy it? Disponível em: http://www.qualitytree.com/feature/biobi.pdf Acessado em: 25/06/2006.

[3] IBM Rational Functional Tester User's Guide, IBM Corporation.

[4] IBM Software - Rational Functional Tester – Product Overview. Disponível em: http://www-306.ibm.com/software/awdtools/tester/functional/. Acessado em: 25/06/2006.

[5] JAKARTA POI - Java API To Access Microsoft Format Files. Disponível em: http://jakarta.apache.org/poi/. Acessado em: 11/09/2006.

[6] KANER, Cem. Improving the Maintainability of Automated Test Suites, 1997. Disponível em: http://www.kaner.com/lawst1.htm. Acessado em: 25/06/2006.

[7] KANER, C.; Falk, J.; Nguyen, HQ. Testing Computer Software. Second Edition, Willey, 1999.

[8] KELLY, Michael. Introduction to IBM Rational Functional Tester 6.1. Disponível em: http://www-128.ibm.com/developerworks/rational/library/04/r-3228/index.html. Acessado em: 25/06/2006

[9] MERCURY QuickTest Professional. Disponível em: http://www.mercury.com/us/products/quality-center/functional-testing/quicktest-professional/ Acessado em: 22/09/2006.

[10] PETTICHORD, Bret. Success with Test Automation, 2001. Disponível em: http://www.io.com/%7Ewazmo/succpap.htm. Acessado em: 25/06/2006.

[11] PRESSMAN, Roger S. Engenharia de Software, Sexta Edição, McGraw-Hill, 2006.

[12] *QA*RUN Product Detail. Disponível em: http://www.compuware.com/products/qacenter/401_ENG_HTML.htm. Acessado em: 20/09/2006.

[13] ROBINSON, Ray. Automation Test Tools Comparison, 2001. Disponível em: http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=2861 Acessado em: 25/06/2006.

[14] SOMMERVILLE, Ian. Engenharia de Software, 6ª Edição, Addison Wesley, 2003.

[15] STEP up the pace of functional & regression testing. Disponível em: http://www.segue.com/products/functional-regressional-testing/silktest.asp. Acessado em: 22/09/2006.

[16] VARGHESE, Jose. Test Automation – An ROI based Approach. Disponível em: http://www.softwaredioxide.com/channels/Content/jose_wipro.pdf. Acessado em: 01/10/2006.

[17] ZAMBELICH, Keith. Totally Data-Driven Automated Testing. Disponível em: http://www.sqa-test.com/w_paper1.html. Acessado em: 25/06/2006.

APÊNDICE A - Comparação entre Ferramentas de Automação de Testes Funcionais

É notável a variedade de opções no Mercado de ferramentas de automação de testes tanto em termos dos tipos das ferramentas de teste que estão sendo oferecidas, quanto do número de vendedores.

A melhor ferramenta para uma situação particular depende do ambiente em que o sistema funciona e da metodologia de teste que será usada, que por sua vez dirá como a automação será usada para suportar o processo.

Este apêndice avaliará as principais marcas de ferramentas de automação baseado nas características de cada ferramenta, irá testar a capacidade de execução, qualidade do mapeamento de objetos da tela, capacidade de integração da ferramenta, suporte a ambiente e extensibilidade. As seguintes ferramentas serão avaliadas *Compuware QARun* [12], *Mercury QuickTest Professional* [9], *Rational Functional Tester* [4] e *Borland Segue SilkTest* [15].

No final deste apêndice será construída uma matriz entre cada ferramenta e as categorias da avaliação, tirada do artigo de Ray Robinson [13]. Esta matriz provê uma maneira rápida e fácil de consultar as características de cada ferramenta de automação de teste. Os critérios para a escolha da ferramenta.

A descrição detalhada de cada categoria utilizada na matriz será dada a seguir.

1. *Record & Playback*

Ao automatizar, esta é a primeira coisa que um profissional de teste faz. Ele vai gravar um script, observar o seu código e reproduzi-lo.

Esta categoria detalha:

a. O quão fácil é gravar e reproduzir um teste
b. Se a ferramenta suporta gravação baixo-nível (movimentos do mouse, localização exata na tela, etc.).
c. O quão fácil é ler um arquivo de script que foi gerado pela ferramenta.

2. Testes *Web*

Cada vez mais a internet está presente nas aplicações corporativas. Desta maneira, as ferramentas de testes devem prover funcionalidades para testes de sistemas com arquitetura cliente/servidor. Para tal, as ferramentas devem prover suporte para tabelas HTML, *frames*, diversas plataformas, *links*, etc.

Os critérios desta categoria são:

a. Existem funções que avisam quando uma página acabou de carregar?
b. Existem funções que permitem esperar até carregar uma imagem?
c. Pode-se testar quando um link é válido ou não?
d. Podem-se testar os dados e as propriedades dos objetos *web*?
e. Existe diferenciação entre os tipos e a localização de objetos?
f. Existe diferenciação entre os campos da página? Como título, corpo, etc.

3. Mapeamento de Objetos

As ferramentas de automação de testes devem saber identificar e manipular objetos na interface com o usuário. A maioria dos objetos produzidos irão se comportar da mesma forma de objetos padrão de tela como: botões, *check boxes*, botões de rádio, listas, *edit boxes* e *combo boxes*.

A ferramenta trabalha bem com estes controles padrões? É possível adicionar controles customizados com novas classes de controle? Estas são algumas questões que serão investigadas nesta categoria ao utilizar as ferramentas.

4. Extensibilidade

Esta seção está relacionada com a possibilidade de extensão da ferramenta. Se a ferramenta não suportar determinada funcionalidade é possível criar uma? Este geralmente é um assunto avançado e requer um bom conhecimento em linguagem de programação.

Algumas ferramentas provêem extensões permitindo criar funções métodos e classes, porém apenas utilizando tipos e funções já pré-definidas pela ferramenta ao invés de permitir a extensão da ferramenta além das suas funcionalidades. Provavelmente um programador vai querer utilizar funcionalidade de outras

naturezas, como bibliotecas do sistema operacional ou componentes disponíveis no mercado.

5. Suporte a Ambiente

A ferramenta suporta a última versão de Java, Oracle, WAP, etc.? Muitas ferramentas permitem criar classes, dll`s, etc. para se comunicar com ambientes não suportados.

Esta é uma importante parte da automação, se a ferramenta não suporta o ambiente da aplicação sob teste isto pode ser um grande problema, e provavelmente será melhor utilizar testes manuais.

6. Integração

Como as ferramentas se integram com outras ferramentas? Isto está se tornando cada vez mais importante hoje em dia. A ferramenta permite executar testes a partir de diversas suítes de teste? Existe alguma integração com ferramentas como o Word, Excel ou de gerenciamento de requisitos?

Quando se está gerenciando um projeto de automação de maior porte a integração com outras ferramentas se torna mais explícito. Como gerenciar os *bugs* encontrados, os testes executados automaticamente e manualmente, e quais testes foram executados e quando, sem ter informações perdidas ou duplicadas? Possuir um conjunto de ferramentas integradas neste momento pode ser um diferencial competitivo para a empresa.

Matriz

Cada categoria na matriz é dada uma avaliação entre 1 e 5.

1 = nenhum suporte.

2 = somente suportado por chamada de APIs ou de um plug-in.

3 = suporte básico apenas.

4 = bom suporte, mas existe um melhor suporte.

5 = suporte satisfatório.

A matriz foi construída usando estes conjuntos de valores para cada critério descrito acima. Para cada ferramenta foi somado um total de pontos para ajudar no

processo da avaliação. Quanto maior a soma de pontos melhor a ferramenta, mas é importante relatar que esta é uma avaliação subjetiva e baseada na experiência do autor utilizando cada uma das ferramentas avaliadas.

	Record & Playback	Teste de *Web*	Mapeamento de Objetos	Extensibilidade	Suporte a ambiente	Integração	Total
QuickTest	5	5	3	4	5	4	26
QARun	5	5	3	4	5	4	26
SilkTest	5	5	3	5	5	5	28
Functional Tester	5	5	3	5	5	5	28

Tabela 5 – Matriz de comparação entre Ferramentas de Automação de Testes Funcionais