FlexTest – Um Framework Flexível para a Automação de Testes

Camila Socolowski[1, 2], André Alarcon[2], André Temple de Antonio[2]

[1]Departamento de Engenharia de Computação e Automação Industrial (DCA)
Universidade Estadual de Campinas (Unicamp) – Campinas, SP – Brasil

[2]CPqD Telecom & IT Solutions / DSB – Diretoria de Soluções em Billing
Campinas, SP – Brasil.

{camilas,aalarcon,andret}@cpqd.com.br

Abstract. *Regression testing is applied to ensure the quality of software across multiple versions. This paper presents a framework designed to automate tests recorded by Selenium tool and to allow greater flexibility in the maintenance of test cases by test analysts with different skills. Moreover, the framework provides some facilitators resources to be used at the conference of the expected results of the execution of test cases. The feasibility of this framework was validated through its application in automated testing of a real software.*

Resumo. *O teste de regressão é aplicado para garantir a qualidade de software ao longo de várias versões. Este artigo apresenta um framework desenvolvido para automatizar testes gravados por meio da ferramenta Selenium e para permitir uma maior flexibilidade na manutenção dos casos de teste pelos analistas de teste com diferentes perfis. Além disso, o framework disponibiliza alguns recursos facilitadores a serem utilizados na conferência dos resultados esperados da execução de casos de teste. A viabilidade desse framework foi validada por meio de sua aplicação em automação de testes de um software real.*

1. Introdução

Após seu desenvolvimento, um produto de software usado na indústria precisa evoluir devido à inclusão de novas funcionalidades requeridas pelo cliente a medida que seus negócios mudam. Além disso, defeitos são encontrados durante o uso do software em produção, provocando a necessidade de manutenção. Diante dessa evolução, é um desafio manter a qualidade de software após diversas versões [Park et. al. 2008].

O teste de regressão é usado para garantir qualidade de software, após diversas versões de um produto de software terem sido geradas durante seu desenvolvimento e manutenção. No entanto, o teste de regressão é muito caro, porque requer muitas execuções de casos de teste e o número de casos de teste aumenta consideravelmente à medida que o software evolui [Elbaum et al. 2002].

Esse artigo abordará as técnicas de automação relacionadas à tecnologia utilizada no processo de automação. Mas recomenda-se que antes de iniciar qualquer projeto de automação, sejam analisados o custo-benefício da aplicação das técnicas

seleção de testes de regressão (STR) para se obter um resultado mais eficaz na seleção dos casos de teste que serão automatizados.

O processo de automação definido nesse trabalho aponta primeiramente para a necessidade de identificar com critério as funcionalidades que devem ser automatizadas. Após a identificação dessas funcionalidades e a seleção de casos de teste, o próximo passo é identificar quais as tecnologias e ferramentas poderiam ser utilizadas para automatizar os testes selecionados, e dentro desse contexto identificou-se a necessidade da criação de um framework para automatizar os testes.

Portanto, nesse artigo é proposto o framework FlexTest que tem como objetivo possibilitar a execução de testes funcionais baseados em plataforma web. A aplicação do framework é apresentada em um estudo de caso realizado em um sistema considerado de grande porte.

Este artigo está organizado da seguinte forma: na Seção 2 são descritos os conceitos básicos de testes de regressão e técnicas de automação de testes, na Seção 3 é apresentado o processo de automação utilizado neste trabalho, na Seção 4 é apresentado o framework FlexTest para automação de testes de software, na Seção 5 é apresentado um estudo de caso em que o framework é aplicado em um sistema real e finalmente na Seção 6 são apresentados a conclusão e as sugestões de trabalhos futuros.

2. Definições e Conceitos Básicos

Nas subseções a seguir, são apresentados os conceitos básicos relacionados a teste de regressão e técnicas de automação de testes.

2.1. Teste de Regressão

O teste de regressão é usado para apoiar as atividades de teste e garantir a obtenção da qualidade apropriada ao longo de diversas versões de software [Park et al. 2008]. Quando componentes novos ou modificados, inseridos em um software, causam defeitos a outros componentes não modificados, pode-se afirmar que o sistema em teste regrediu [Binder 2000]. Por isso, os testes aplicados ao novo sistema, contendo as alterações, são chamados “testes de regressão”, que visam evitar a regressão do sistema. As principais técnicas para teste de regressão são: seleção de testes de regressão (STR) [Rothermel and Harrold 1996] e priorização de testes de regressão [Elbaum et al. 2002].

2.2. Técnicas de Automação de Testes

As principais técnicas de automação de teste apresentadas na literatura são: *record & playback*, *scripts*, *data-driven* e *keyword-drivers*.

A técnica *record & playback* consiste em utilizar uma ferramenta de automação de testes para gravar as ações executadas pelo usuário, que interage com a interface gráfica do sistema e converte as ações em scripts de teste que podem ser executados quantas vezes forem necessárias. Essa técnica é considerada simples e prática [Fewster 1999] [Fewster 2001] [Hendrickson 1998].

A técnica de *scripts* é considerada como 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. Em comparação com a técnica *record & playback,* a técnica de *scripts* apresenta maior taxa de reutilização, maior tempo de vida, melhor manutenção e maior robustez dos scripts de teste [Tervo 2001].

A técnica *Data-Driven* 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 armazenar 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 [Nagle 2000] [Zambelich 1998].

A técnica *Keyword-Driven* 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 [Fewster 1999] [Nagle 2000].

As ferramentas que fornecem suporte aos testes de sistema geralmente integram uma ou mais destas técnicas de automação permitindo sua utilização na execução de vários tipos de testes de sistema, como os testes funcionais, os testes de regressão e os testes de performance e etc.

3. Processo de Automação de Teste

O processo de automação definido e aplicado no CPqD foi dividido em algumas fases como identificação do que deve ser automatizado, definição dos casos testes e resultados esperados e identificação da tecnologia que será utilizada na automação dos casos de teste. O processo de automação foi definido em algumas fases:

3.1. Identificação do escopo da automação

Esta fase define quais são as funcionalidades que devem ser automatizadas. Uma das fases mais importantes no processo de automação é a decisão do que se deve automatizar. Durante a execução dessa fase foram analisados os dados históricos relacionados à abertura de defeitos pelo cliente na homologação e produção do sistema. Alguns passos importantes foram dados nessa etapa:

- Análise, seleção e sumarização dos defeitos de regressão abertos pelos clientes em homologação e/ou em produção por funcionalidade;
- Definição das funcionalidades que serão automatizadas.

3.2. Identificação dos casos de teste

Esta fase define quais os casos de teste devem ser automatizados da funcionalidade que foi selecionada na fase anterior. Nessa fase, foi necessário identificar os casos de teste que poderiam ser do fluxo básico e/ou alternativo, e os pontos de verificação (oráculos)

para cada caso de teste. Um caso de teste pode apresentar mais de um oráculo a ser validado.

3.3. Identificação da tecnologia

Esta fase define a forma como os oráculos definidos na fase anterior serão validados e quais serão as tecnologias que deverão ser utilizadas para essa validação. Nesse passo, foi realizado um estudo de técnicas de automação e ferramentas de automação.

Após análise dos oráculos para cada caso de teste, verificou-se a necessidade de realizar algumas verificações que não eram atendidas pela ferramenta *Selenium*, que utiliza a técnica "*record & playback*". Com isso, surgiu a necessidade da criação de um framework que permitisse verificações mais complexas, como por exemplo, verificações em banco de dados, que se integrasse com a ferramenta Selenium, que fosse mais flexível para inclusão de novos tipos de verificações e que permitisse a manutenção mais fácil e ágil dos casos de testes gravados.

4. Framework FlexTest

4.1. Arquitetura do Framework

O framework FlexTest é apresentado na Figura 1 em um diagrama que exibe os seus componentes. O núcleo principal do sistema é o Controle de Execução (CE) que é responsável por disparar e controlar todo o processamento de uma planilha de automação de testes. Ao ser iniciado, o CE gera um arquivo de *log* que irá conter as informações dessa execução. O primeiro passo a ser realizado é a identificação dos arquivos do tipo CSV que estão armazenados na pasta de entrada de dados. Para cada arquivo o CE irá criar um log específico e iniciará o *parser* de cada um desses arquivos.

O componente *parser* é capaz de interpretar uma sequência de comandos de um arquivo no formato CSV e preparar os comandos para serem executados pela componente CE. Algumas regras foram criadas para facilitar o preenchimento e entendimento do arquivo como, por exemplo, inclusão de comentários e continuação de um comando na próxima linha, a fim de facilitar a identação. Ao finalizar o *parser* dos arquivos, os comandos e parâmetros correspondentes de cada arquivo CSV são retornados para o CE.

Um arquivo de configuração irá conter todos os possíveis comandos do framework bem como o respectivo caminho para cada implementação. Caso ocorra qualquer problema no comando identificado pelo *parser*, o CE imediatamente aborta a execução do CSV em processamento. Caso contrário, os passos de validação e execução de cada comando serão disparados. A validação tem como objetivo identificar possíveis problemas de sintaxe nos comandos e parâmetros antes da execução dos comandos no arquivo CSV.

Uma validação não efetiva da sintaxe dos comandos pode permitir que uma eventual falha só apareça durante a execução dos testes automatizados, e isso acarretaria em prejuízo para os testes. Nesse caso é interessante antecipar ao máximo a identificação de problemas nos comandos do arquivo CSV. A execução dos comandos é o último passo a ser gerenciado pelo CE de acordo com a funcionalidade de cada comando.

Uma das características interessantes do framework é a facilidade com que um novo comando pode ser disponibilizado em sua estrutura. Para isso, o novo comando deve ser implementado em uma classe Java, contendo os serviços de validação e execução e incluído no arquivo de configuração. Todas as tarefas de chamadas ao novo comando, *log*, e outras, são atribuídas ao framework e ficam transparentes à implementação dos comandos.

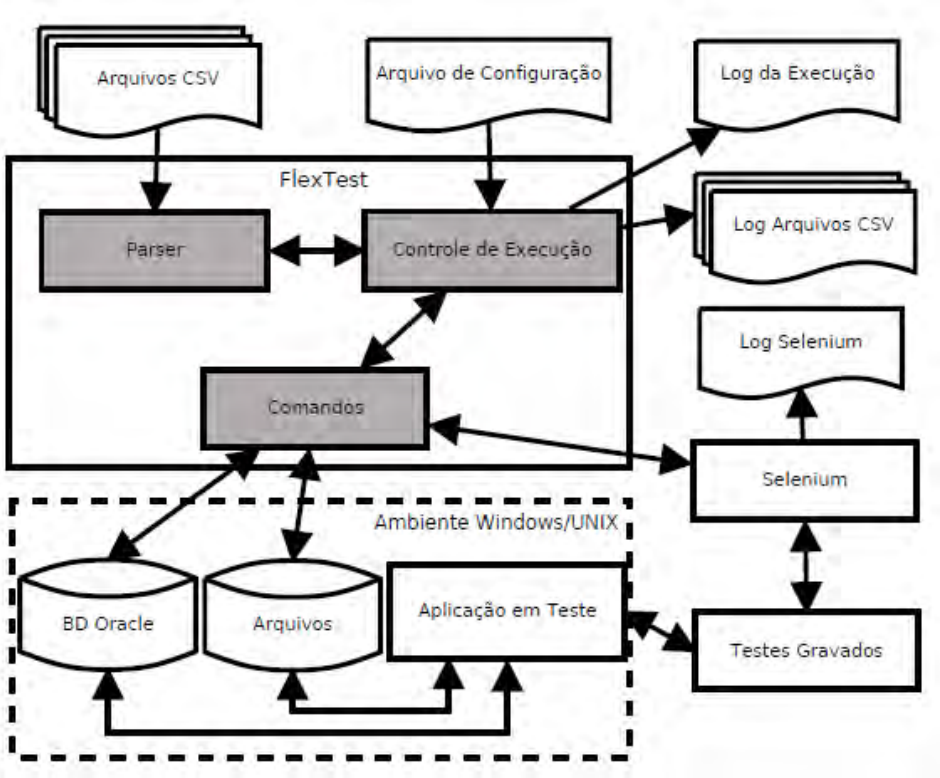

Figura 1 – Arquitetura Completa do Framework FlexTest

Um dos comandos disponibilizados pelo framework executa testes da ferramenta de automação Selenium gerando um *log* específico. Essa ferramenta de automação de teste além de suporte nativo a técnica '*record & playback*', oferece a possibilidade de exportar os casos de teste gravados para uma linguagem de programação, como por exemplo Java, o que a torna bastante compatível com a arquitetura do framework FlexTest.

O framework possui outros comandos como verificação de SQL, comparação entre arquivos, import e export de banco, etc.

4.2. Funcionamento do Framework

O arquivo CSV que é utilizado como entrada para o framework contém um conjunto de passos (comandos) que serão executados para cada teste. Por exemplo, importar determinado banco de dados, executar um teste específico gravado por meio da ferramenta *Selenium* e executar uma lista de consultas em banco.

Os testes gravados por meio do *Selenium* são portáveis e incrementais. Para adicionar um novo caso de teste ao framework (teste gravado pelo Selenium), o analista de teste deve incluir o código Java gerado pelo Selenium em um arquivo,em um diretório especifico que está configurado em sua máquina e clicar no ícone *build*.

O framework disponibiliza também um comando flexível para conferência dos dados no banco. Como todo teste automatizado verifica um resultado esperado, o usuário deverá incluir como parâmetro desse comando as consultas e os resultados esperados para determinado teste. O framework, no momento da execução desse comando, se encarrega de acessar o banco de dados, executar as consultas desejadas e comparar os resultados obtidos com os resultados esperados, incluindo essas informações no *log* de execução do arquivo CSV em teste.

5. Estudo de caso – Utilização do Framework FlexTest na Automação de um Sistema de Gestão de Telecomunicações

5.1. Características do Sistema

A validação do framework foi realizada através da sua aplicação na automação de testes de um sistema de faturamento e de um sistema de gestão de recursos de telecomunicações (GRT) da Fundação CPqD.

O sistema apresenta aproximadamente 300.000 linhas de código e se aplica ao mercado de telecomunicações e sua especificação funcional engloba regras de negócios complexas para empresas operadoras de telecomunicações.

5.2. Fases do Processo de Automação no Sistema de Faturamento e no Sistema GRT

O processo de automação no sistema de faturamento e no GRT passou pelas fases de identificação do escopo, identificação dos casos de teste e identificação das tecnologias descritas na seção 3 desse artigo.

Foram selecionados para iniciar o processo de automação 40 funcionalidades, 250 casos de testes e o framework como ferramenta de automação de testes. As gravações foram realizadas utilizando a ferramenta *Selenium* que possibilitou a exportação do código Java que foi incorporado dentro dos casos de teste do framework.

Após a gravação e a inclusão dos casos de teste no framework, foi criado o arquivo CSV que direcionou a execução dos testes e as conferências que foram realizadas.

5.3. Resultados Obtidos

Nesta seção são apresentados os resultados obtidos na aplicação da automação de teste no sistema de faturamento e no sistema GRT.

Durante todo o processo de automação pode-se observar a facilidade que os analistas de teste com diferentes perfis tiveram para gravar e/ou alterar casos de teste existentes incluindo-os com facilidade na execução do framework. Isso foi possível observar, pois o tempo real gasto nas gravações, alterações de scripts e execuções com sucesso do teste automatizado foi inferior ao tempo estimado para essas mesmas

atividades, em experiências anteriores de automação, quando outras ferramentas eram utilizadas e era necessária a intervenção de um implementador para avançar na automação dos testes.

Foi possível observar a facilidade de verificação de alguns resultados esperados para alguns testes mais complexos de serem validados na automação por meio de comandos SQL que comparam o valor retornado do banco com o valor esperado pelo analista de teste e por fim registram no *log* se o resultado da consulta está correto ou incorreto de acordo com os parâmetros informados.

6. Trabalhos Relacionados

Existem outros trabalhos que abordam ferramentas de suporte à automação de teste de software, porém estas ferramentas apresentam limitações em suas funcionalidades. O framework FlexTest foi criado para atender a uma premissa básica de facilidade, tanto no sentido de ser utilizada por diferentes perfis de analistas de teste, como a de permitir a inclusão de novas funcionalidades com maior agilidade, como também permitir facilidades na verificação de resultados esperados e flexibilidade nas configurações.

Uma grande parte das ferramentas de automação de teste disponíveis são voltadas para a técnica *record & playback*. As ferramentas Functional Tester, WinRunner, QuickTestPro, TestSmith e SilkTest, além da funcionalidade de gravação de scripts de teste, oferecem uma linguagem de programação permitindo que a técnica de programação de scripts seja utilizada.

No trabalho de [Fantinato et. al, 2004] é apresentado um framework que não podia ser utilizado para sistemas baseados na plataforma web, não possibilitava que o processo de automação fosse assumido totalmente pelo analista de teste sem conhecimento em linguagens de programação e solicitava o preenchimento de muitos dados de teste nas planilhas, o que aumentava o tempo de execução dessa atividade.

O framework FlexTest foi criado para fornecer um conjunto maior de suporte a automação de testes. A ferramenta gratuita *Selenium* é um dos componentes deste framework. Um dos objetivos do framework é possibilitar a execução de testes para sistemas baseados em plataforma web.

7. Conclusão e Trabalhos Futuros

Esse trabalho apresenta um framework que foi criado e utilizado no processo de automação do sistema de faturamento da Fundação CPqD para o mercado de telecomunicações. Observou-se que para obter um resultado mais satisfatório na automação dos testes é necessário considerar a tecnologia que será utilizada, mas é necessário também considerar outras questões como a seleção das funcionalidades e dos casos de teste que devem ser automatizados.

Algumas variáveis como o custo alto da automação, manutenção dos casos de teste automatizados, falta de flexibilidade, diferentes perfis de analistas de teste envolvidos no processo de automação, foram consideradas na especificação do framework e na definição das fases do processo de automação.

Como trabalho futuro pretende-se estender o framework FlexTest para realizar comparações entre outros tipos de arquivos, localizar e comparar alguns itens dentro do

arquivo e incluir novos comandos a serem utilizados pelos analistas de teste. Além disso, pretende-se aplicar o processo de automação em outros sistemas e coletar uma quantidade maior de informações relacionadas ao processo de automação.

8. References

Park H., Ryu, H., and Baik, J.(2008). Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing. Ssiri, 0:39-46.

Elbaum,S., Malishevsky, A.,and Rothermel, G.(2002). Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, 28(2):159-182.

Kim,J.,-M. and Porter, A.(2002). A history-based test prioritization technique for regression testing in resource constrained environments. In ICSE 2002: Proceedings of the 24th Internacional Conference on Software Engineering, pages 119-129. New York, NY, USA.ACM.

Binder, R. V. (2000). Testing Object-Oriented Systems: Models, Patterns and Tools. Addison-Wesley Longman.

Rothermel G. and Harrold, M.J.(1996). Analyzing regression test selection techniques. IEEE Trans. Softw. Eng. 22(8): 529-551.

Fewster, M. & Graham, D., “Software Test Automation”, Addison-Wesley, 1999.

Fewster, M., “Common Mistakes in Test Automation”, Proceedings of Fall Test Automation Conference, 2001.

Hendrickson, E., “The Differences Between Test Automation Success And Failure”, Proceedings of STAR West, 1998.

Tervo, B., “Standards For Test Automation”, Proc. of STAR East, 2001.

Nagle, C.,“ Test Automation Frameworks”, disponível em http://members.aol.com/sascanagl/DataDrivenTestAutomationFrameworks.htm, 2000.

Zambelich, K., “Totally Data-driven Automated Testing”, disponível em http://www.sqatest.com/w_ paper1.html, 1998.

Fantinato, M., Cunha, A., Dias, S., Mizuno, S., and Cunha, C. (2004). AutoTest–Um Framework Reutilizável para a Automação de Teste Funcional de Software. Simpósio Brasileiro de Qualidade de Software.

CPqD. Disponível em http://www.cpqd.com.br.

Functional Tester. em; http://www-01.ibm.com/software/awdtools/tester/functional.