Graduação em Ciência da Computação

“TaRGeT Scripts Generation: Um plug-in de geração automática de scripts de teste ”

Por

Hugo Luís de França Siqueira

Trabalho de Graduação

Universidade Federal de Pernambuco
www.cin.ufpe.br

RECIFE, DEZEMBRO/2010

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

HUGO LUÍS DE FRANÇA SIQUEIRA

"TaRGeT Scripts Generation: um plug-in de geração automática de scripts de teste"

ESTE TRABALHO FOI APRESENTADO À GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: PAULO HENRIQUE MONTEIRO BORBA

RECIFE, 2010

AGRADECIMENTOS

A realização de uma etapa importante na vida de uma pessoa nunca é solitária, nossos objetivos são alcançados com a ajuda de várias pessoas que passaram por nossas vidas e nos ajudaram de alguma forma.

Gostaria de agradecer a Deus por ter colocado uma mãe responsável e preocupada com a educação de seu filho. Graças ao esforço dela consegui estudar em bons lugares e assim ter a possibilidade de concorrer a uma vaga de um curso tão disputado. A Fá que nos apoiou em vários momentos da nossa vida e me acompanhou como pai.

Aos meus colegas de curso que infelizmente não estreitei laços, agradeço por terem compartilhado conhecimento e me ajudado diversas vezes. Aos monitores, principalmente nos períodos iniciais, Sylvinha e Diana.

À minha família que durante todos esses anos esperou por esse momento. Daisy, Titia Lourdinha, Neide, todas as tias e primos.

Aos meus amigos que durante todo esse período me incentivaram a finalizar o curso, após um período de cansaço mental. Bruno, Aguinaldo, Marlon, Júnior, Nildinho, Ana Paula (lourinha), Hugo de Lima, todos vocês não estarão presentes somente aqui, mas na minha história toda. Agradeço pelo apoio que todos vocês me deram e me dão. A bruninho, meu sobrinho cachorro, pelas madrugadas de companhia.

À Michelle e Laís que me ajudaram com dúvidas relativas à TaRGeT e tiveram paciência durante o período de desenvolvimento. Aos companheiros de trabalho da Qualiti, em especial a Felipe Farias, Felipe Omena, Oromar Dantas e Luís Filipe Pessoa.

Ao Professor Paulo Borba por ter acreditado no meu potencial de trabalho e ter me encaminhado à Qualiti, onde conheci pessoas importantes à minha formação profissional.

TARGET SCRIPTS GENERATION – UM PLUG-IN DE GERAÇÃO AUTOMÁTICA DE SCRIPTS DE TESTE

Autor: Hugo Luís de França Siqueira
Orientador: Paulo Henrique Monteiro Borba

RESUMO

A disciplina de testes, na engenharia de software, surgiu da necessidade de garantia de qualidade dos produtos desenvolvidos. Processos foram criados para sistematizar as atividades e gerenciar melhor os artefatos gerados. Com a crescente busca pela qualidade dos softwares implementados houve um impulso no sentido de melhorar os processos de teste por parte da indústria e da academia, que vêm investindo na criação de sistemas que otimizam esses processos. Ferramentas de automação trazem ganhos à etapa de execução, por permitir que a mesma seja realizada de forma mais rápida e consumindo menos recursos se comparada com a execução manual. Porém, a técnica de automação *Record and Playback* apresenta problemas quando casos de testes diferentes possuem passos em comum, devido à necessidade de gravação redundante desses passos. Com o intuito de evitar que passos iguais de diferentes casos de teste sejam gravados repetidamente, o objetivo deste trabalho é criar um plug-in para a TaRGeT que permita o compartilhamento dos *scripts* gravados, diminuindo o tempo de automação das suítes e propiciando o início antecipado da execução.

Palavras-chave: Testes de Software. Automação de testes. Geração de *scripts*

Author: Hugo Luís de França Siqueira
Adviser: Paulo Henrique Monteiro Borba

ABSTRACT

The test discipline on the software engineer came from the need for quality assurance of the developed products. Processes were created to systematize the activities and to manage the generated artifacts better. With the growing seek for quality of the implemented systems, there was a great interest in to become the test processes better by part of both industry and academy, which are investing on the creation of systems that optimize these processes. Automation tools bring gains to the execution phase because they allow a faster realization and less consuming resources if compared to manual execution. However, the Record and Playback automation technique present problems when different test cases have common steps, due to the redundant need for recording these steps. With the goal of avoiding that equal steps from different test cases be recorded repeatedly, the objective of this work is create a plug-in for the TaRGeT system which allow the sharing of recorded scripts, reducing the automation time of the suites and providing an anticipated execution beginning.

Keywords: Software Testing. Test Automation. Scripts Generation

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta uma visão geral desta monografia. A Seção 1.1 apresenta a motivação deste trabalho. A Seção 1.2 demarca o escopo deste trabalho bem como as contribuições esperadas. Finalmente, a Seção 1.3 fornece uma visão dos capítulos deste documento.

1.1 Motivação

Desde o final dos anos 70, com a eclosão da crise do software, a comunidade científica começou um movimento no sentido de melhorar a forma como os sistemas eram construídos e assim evitar que os prazos e custos dos projetos “estourassem”, como também garantir a qualidade do produto entregue. (1)

Ao longo dos anos, as aplicações computacionais passaram a ficar cada vez mais complexas e era necessário que os processos de desenvolvimento garantissem a qualidade do produto. Para atender a este requisito, a disciplina de testes surgiu. Inicialmente os testes eram realizados sob demanda e, mesmo havendo esta etapa no ciclo de desenvolvimento, ela precisava ser mais robusta.

Processos de teste foram desenvolvidos com o propósito de sistematizar as atividades e deixá-las mais organizadas, como acontecia em outras engenharias, surgindo então o termo “Engenharia de Testes”, que passou a ganhar uma maior importância no ciclo de desenvolvimento de software e vem recebendo grandes investimentos da indústria e da academia no sentido de desenvolver novas técnicas que a torne mais eficaz. (2)

A automação de testes desponta, então, como uma forma de tornar o processo de execução mais rápido, aumentando também a quantidade de cenários testados quando o tempo de execução é limitado. Por ser uma atividade cara, as empresas que automatizam seus testes nem sempre têm retorno financeiro direto num curto espaço de

tempo, a economia é atingida a médio prazo, onde os cenários já automatizados chegam a custar 80% menos em comparação com a execução manual. (3)

Diante das diversas técnicas para automação de testes, a de *Record and Playback*, na qual o cenário de teste é gravado e um *script* de execução é criado, apresenta ineficiências quando os fluxos a serem gravados são semelhantes, ou seja, se *n* fluxos possuem *k* passos iguais, deve-se gravar os *k* passos *n* vezes, resultando num consumo maior de esforço e tempo. Como, em projetos de teste, é bastante comum a ocorrência de cenários que possuem passos iguais e a técnica de gravação de *scripts* obriga que tais fluxos sejam gravados sem reuso, seria interessante adaptar a técnica de forma que houvesse um reaproveitamento dos *scripts* já criados.

O objetivo deste trabalho é criar um produto responsável por armazenar os *scripts* gerados para cada passo do teste e reutilizá-los para casos de teste que possuam passos em comum. Espera-se que este reuso reduza o tempo de automação, impactando, conseqüentemente, no tempo de entrega dos projetos.

1.2 Escopo e Contribuições Esperadas

A TaRGeT (*Test and Requirements Generation Tool)* é uma ferramenta utilizada para gerar casos de testes baseados nas combinações e reuso de passos de um documento formal de caso de uso. (4) Reaproveitando este conceito de reuso da TaRGeT, o produto tratado neste trabalho consiste de um *plug-in* que visa gerenciar a automação dos testes gerados na ferramenta.

Desenvolvido em conjunto com uma equipe de uma fábrica de testes interessada na solução, a metodologia de desenvolvimento se baseou, primeiramente, na escolha de duas ferramentas para gravação dos *scripts*: O *Selenium* e o *Rational Functional Tester (RFT)*, que foram analisadas no sentido de prover em sua estrutura formas de reusar os *scripts* gravados. Para a realização desta etapa, foi criado um documento de caso de uso que foi submetido à TaRGeT, para a geração automática dos casos de teste. Alguns

desses passos foram gravados nas ferramentas citadas e uma junção manual dos *scripts* foi feita, para analisar o comportamento do código resultante. Em ambas as ferramentas os *scripts* executaram sem falhas e o esforço de integração das partes, realizada manualmente, foi mapeado para ser executado de forma automática pelo *plug-in*. O método de avaliação da solução consistiu em gravar novos passos de teste e executar os *scripts* gerados pelo *plug-in*, com o intuito de validar o seu correto funcionamento. Avaliação do impacto no tempo de automação do processo de teste está inserido como trabalhos futuros.

Espera-se ao final do trabalho que a TaRGeT, através do plug-in, disponibilize uma *feature* que impacte no tempo de execução das suítes e a complemente como ferramenta de apoio ao processo de testes, passando a otimizar não só a etapa de arquitetura como também a de execução.

1.3 Estrutura da Monografia

O restante deste trabalho está organizado da seguinte maneira:

O Capítulo 2 apresenta uma visão geral sobre testes de software. Uma revisão dos conceitos e atividades mais comuns do processo de teste são explanadas.

O Capítulo 3 está voltado para a automação de testes, descrevendo as técnicas mais utilizadas, além de vantagens, desvantagens e limitações.

O Capítulo 4 foca no desenvolvimento de *plug-ins*, nas plataformas disponibilizadas pelo Eclipse e seus principais componentes, além de apresentar a TaRGeT, mostrando sua arquitetura e principais funcionalidades.

O Capítulo 5 aborda detalhes do plug-in desenvolvido, funcionalidades criadas, análises das ferramentas escolhidas para gravação dos *scritps*, além do fluxo de trabalho utilizando a TaRGeT.

O Capítulo 6 descreve as atividades realizadas para a avaliação do produto.

O Capítulo 7 apresenta a conclusão deste trabalho onde são descritas as principais dificuldades encontradas, as contribuições e também as perspectivas de trabalhos futuros.

2 TESTES DE SOFTWARE

Este capítulo tem como objetivo dar uma Visão Geral sobre os conceitos relativos a Testes de Software. Na seção 2.1 é apresentada uma visão geral sobre a importância da atividade de teste. Na seção 2.2, são mostradas as diferentes abordagens de teste. Na seção 2.3 são descritos os tipos de teste. Na seção 2.4 os níveis ou fases de teste são revisados. Na seção 2.5, o processo de teste é tratado e atividades básicas são mostradas. Por fim, na seção 2.6 são apresentadas as considerações finais.

2.1 Visão Geral

Como uma disciplina emergente na engenharia de software, os processos de testes vêm ganhando destaque devido à necessidade de garantia de qualidade dos produtos, que têm estado cada vez mais complexos e presentes nas mais diversas esferas do conhecimento.

Inseridos fortemente nas atividades sociais e econômicas, os sistemas computacionais possuem um importante papel no que diz respeito à agregação de valor aos produtos e serviços aos quais se propõem a dar suporte. Isto obriga que estes softwares atendam a um mínimo de qualidade, para que, em ambiente de produção, não haja ocorrência de falhas e conseqüentemente perda de confiabilidade do produto entregue. (2)

O presente capítulo tem como objetivo revisar os conceitos de testes de software necessários ao pleno entendimento do problema aqui proposto. A seção seguinte expõe as principais abordagens de teste, útil para entender onde o produto será aplicado.

2.2 Abordagens de teste

Para realizar a atividade de teste, é preciso primeiramente saber qual abordagem será tomada, pois, dependendo da escolha, diferentes técnicas são aplicadas, objetivando a criação de casos de testes significativos, para garantir a máxima cobertura das situações passíveis de falha. As abordagens largamente conhecidas na indústria são: a funcional (caixa-preta) e a estrutural (caixa-branca). A abordagem estrutural, em linhas gerais, refere-se ao teste realizado na estrutura interna do sistema, ou seja, o código e suas entidades elementares como métodos e classes são avaliados.

Como a TaRGeT gera casos de teste a partir de um documento de funcionalidades do sistema (caso de uso), a abordagem funcional será detalhada a seguir:

Abordagem funcional ou caixa-preta

O arquiteto de testes, de posse do documento de especificação de requisitos, modela os cenários de teste, baseando-se nos requisitos funcionais e não funcionais do sistema. Nesta abordagem nenhum detalhe do funcionamento interno do sistema é levado em consideração, o que se testa é a correta execução dos requisitos levantados e mapeados em casos de uso. Para a abordagem funcional, as técnicas descritas pelo Syllabus (5) são as seguintes:

- Partição de equivalência
- Análise do valor limite
- Tabela de decisão
- Teste de transição de estados
- Teste de caso de uso

Dentro desta classificação, a técnica de teste de caso de uso é a que está dentro do escopo deste trabalho.

Teste de caso de uso

Um documento de caso de uso descreve as funcionalidades do sistema, baseado nas interações entre os atores (usuários ou outros sistemas), nas precondições e pós-condições após a realização daquele caso de uso.

Por definir passos a serem realizados pelo sistema e possuir os resultados esperados na sua definição, os casos de uso são uma fonte extremamente útil para extrair casos de teste e assim validar, de uma forma mais aderente às necessidades do cliente, se o sistema está de acordo com o que foi especificado.

2.3 Tipos de teste

A disciplina de teste é bastante abrangente e possui atividades de verificação e validação que são aplicadas para testar diferentes aspectos e situações inerentes a um sistema computacional. São definidos vários tipos de testes com objetivos específicos, a fim de garantir que o sistema está dentro dos requisitos de qualidade levantados. Os principais tipos, segundo (6), são:

- Teste Funcional
- Teste de Recuperação de falha
- Teste de Segurança e controle de acesso
- Teste de integridade de dados
- Teste de performance
- Teste de volume
- Teste de estresse

- Teste de configuração
- Teste de instalação
- Teste de interface
- Teste de documentação
- Teste de ciclo de negócio
- Teste de regressão

A seguir uma revisão mais detalhada dos tipos mais importantes para o contexto do trabalho.

Teste Funcional

Neste tipo de teste os requisitos funcionais do sistema são postos sob testes, por meio de cenários que cobrem os fluxos de execução das funcionalidades do sistema. O teste funcional verifica o programa, validando seu funcionamento tendo como base algum documento de arquitetura ou especificação. (5)

A técnica de teste de caso de uso da abordagem caixa-preta, vista anteriormente, se encaixaria perfeitamente neste tipo de teste, pois tem como alvo para a modelagem dos cenários o documento de caso de uso do sistema, ou seja, o próprio documento de funcionalidades da aplicação.

Este tipo de teste é o que a TaRGeT se propõe a realizar, através da geração de cenários por meio de um documento de caso de uso.

Teste de Regressão

Classificado como teste relacionado à mudanças, de acordo com o Syllabus (5), o teste de regressão é realizado quando uma alteração no sistema sob teste é feita, para verificar se estas novas modificações inseriram no sistema erros inicialmente não

presentes. Este teste consiste em re-executar cenários que já foram criados e assim validar que as funcionalidades anteriores não apresentam erros devido à inserção de novas *features*. (7)

Será visto no capítulo 3 que a automação é bastante interessante para a realização de testes de regressão, pois a execução de *scripts* gravados ao longo do ciclo de desenvolvimento de um sistema é feita de forma bastante eficiente.

2.4 Fases de teste

Dependendo do grau de evolução do sistema desenvolvido, têm-se diferentes fases ou níveis de teste que podem ser aplicados aos artefatos criados.

O Syllabus (5) define quatro níveis de testes:

- Teste de componente
- Teste de integração
- Teste de sistema
- Teste de aceitação

Em fases iniciais da implementação, testes de componentes são realizados, validando as entidades de mais baixo nível (classes, métodos). Em seguida elementos que trocam informações têm sua interação testada nos testes de integração. Posteriormente, no teste de sistema, o funcionamento da aplicação como um todo é verificado e por fim, no teste de aceitação, o cliente valida, utilizando o *software*, se as funcionalidades estão de acordo com suas expectativas.

No teste de sistema são realizados testes funcionais, alvo da TaRGeT e do produto aqui proposto. Um dos objetivos do teste de sistema é simular o ambiente real

de produção e com isso desvendar possíveis problemas que aconteceriam se esses softwares fossem postos em execução no cliente. (5)

A próxima seção trata dos principais conceitos de processo de testes. O intuito é apresentar as diferentes etapas da atividade, com o objetivo de esclarecer os pontos que a ferramenta proposta pretende trazer contribuições.

2.5 Processo de Teste: conceitos básicos

A indústria, de uma forma geral, visando atingir metas de qualidade e otimizar sua forma de produção, passou a desenvolver métodos sistemáticos de trabalho que, quando aplicados de maneira correta, trazem resultados mais significativos em termos de qualidade. Estes métodos, ao longo do tempo, tornaram-se atividades bem documentadas, com papéis definidos e formaram a base para a definição do que hoje conhecemos como processos.

Processos bem definidos estão presentes em qualquer ramo da engenharia e servem para guiar as atividades do processo produtivo das organizações.

Com a concepção do termo “Engenharia de Software” e da crescente necessidade de produtos mais confiáveis no mercado, alavancada por uma demanda maior por softwares de alta complexidade, processos para a engenharia de software também foram elaborados, com o intuito de guiar uma atividade realizada desordenadamente, cujos projetos “estouravam” prazo e custo e o produto entregue estava aquém do esperado.

Atualmente a engenharia de software é composta de diversos processos presentes nas diferentes disciplinas do ciclo de vida de um sistema. Há atividades para a disciplina de requisitos, análise e projeto, desenvolvimento, entre outras. A disciplina de testes, apesar de mais recente, também contempla um conjunto de tarefas que formam o processo de testes.

Segundo (5), o processo básico de teste consiste das seguintes fases:

- Planejamento e controle
- Análise e modelagem
- Implementação e execução
- Avaliação do critério de saída e relatórios
- Atividades de encerramento de testes

Cada uma dessas fases possui atividades específicas, desempenhadas por papéis definidos, gerando artefatos de saída que serão utilizados em fases subseqüentes ou mesmo em outros processos da engenharia de software.

É importante salientar que as atividades descritas para o ciclo básico do processo de testes só acontece, mesmo sendo elementar, em empresas que já possuem um certo grau de maturidade no seu processo de desenvolvimento. Modelos de maturidade funcionam como um guia para a implantação de processos melhores e mais robustos, indicando atividades que devem ser realizadas para prover um processo de maior qualidade.

Dentre as atividades básicas, as que apresentam mais possibilidades de melhoria são as etapas de “Análise e modelagem” e “Implementação e execução”, e o artefato desenvolvido tem como objetivo otimizar esta última fase. A seguir elas serão detalhadas, a fim de que um melhor entendimento do problema a ser resolvido seja alcançado.

Análise e Modelagem

Após todo o planejamento do ciclo de teste, deve-se proceder com a análise e modelagem dos testes, baseando-se nos documentos de requisitos e arquitetura do sistema. A partir destes documentos é possível aplicar técnicas adequadas para a criação

dos cenários de testes, visando cobrir situações específicas e passíveis de falha da arquitetura do software.

Questões relativas ao ambiente de teste, solicitação de massa de dados precisam ser estabelecidas nesta fase, para que os casos de testes modelados possam estar completos para seguir para a etapa de execução.

O Syllabus (5) define como atividades desta etapa:

- Revisar a base de testes (requisitos, arquitetura, modelagem, interfaces).
- Avaliar a testabilidade dos requisitos do sistema.
- Identificar condições ou requisitos de testes e dados de testes baseados na análise dos itens de teste, na especificação, no comportamento e na estrutura.
- Projetar e priorizar os casos de testes.
- Identificar a necessidade de dados para teste suportando as condições e casos de testes.
- Planejar a preparação do ambiente de teste e identificar a infraestrutura e ferramentas necessárias.

A etapa de análise e modelagem, dentro do processo de teste, é a que se mostra mais crítica, devido a sua importância para o restante do processo. Se esta etapa é realizada de forma negligente, a cobertura dos testes é prejudicada, acarretando, conseqüentemente, em perda de qualidade. Por esta razão, a fase é alvo da academia e da indústria no que tange a sua constante melhoria através não só da inserção de novas atividades que a torne mais eficaz, como também no desenvolvimento de ferramentas de apoio.

Implementação e execução

Após todo o esforço na criação dos cenários de teste, identificação de massa de dados e estruturação do ambiente de teste, tem-se início a fase de implementação dos cenários e execução do ciclo.

É nesta fase que todas as técnicas empregadas na fase de modelagem são postas em prática, a fim de encontrar defeitos que passaram despercebidos pela equipe de desenvolvimento.

A execução dos testes pode ser feita de forma manual ou automática. Manualmente, o testador executa passo a passo os cenários elicitados na fase de modelagem e verifica se os resultados esperados estão de acordo com a resposta do sistema. O processo manual de execução é mais lento e custoso, o que faz com que esta etapa também seja um ponto bastante visado para implantação de técnicas e ferramentas que a torne mais eficaz.

A automação de testes surgiu da necessidade de tornar o processo manual utilizável somente em situações nas quais a criação de *scripts* não fosse possível, ou seja, era necessário diminuir o tempo do ciclo de testes e com o advento das técnicas de automação e das ferramentas, os casos de testes passaram a ser mapeados em *scripts*, o que facilitou a realização de testes de regressão e redução de tempo do ciclo de execução. Detalhes sobre automação serão descritos no capítulo 3.

A seguir as atividades listadas pelo Syllabus (5) para esta fase:

- Desenvolver, implementar e priorizar os casos de teste.
- Desenvolver e priorizar os procedimentos de teste, criar dados de teste e, opcionalmente, preparar o ambiente para teste e os *scripts* de testes automatizados.
- Criar suítes de teste a partir dos casos de teste para uma execução de teste eficiente.
- Verificar se o ambiente está preparado corretamente.

- Executar os casos de testes manualmente ou utilizando ferramentas de acordo com a seqüência planejada.
- Registrar os resultados da execução do teste e anotar as características e versões do software sob teste, ferramenta de teste e testware[1].
- Comparar resultados obtidos com os resultados esperados.
- Reportar as discrepâncias como incidentes e analisá-los a fim de estabelecer suas causas (por exemplo, defeito no código, em algum dado específico de teste, na documentação de teste ou uma execução inadequada do teste).
- Repetir atividades como resultado de ações tomadas para cada discrepância. Por exemplo, re-execução de um teste que falhou previamente quando da confirmação de uma correção (teste de confirmação), execução de um teste corrigido e/ou execução de testes, a fim de certificar que os defeitos não foram introduzidos em áreas do software que não sofreram modificações, ou que a correção do defeito não desvendou outros (teste de regressão).

Nota-se que esta etapa também se mostra crítica para o ciclo de teste, pois é por meio dela que se evidenciam os problemas dos sistemas desenvolvidos e uma redução na qualidade da execução desta fase pode render diversos prejuízos, tanto em termos de retrabalho, atraso na validação do *release*, mas principalmente na perda da credibilidade do produto entregue.

[1] Termo utilizado para o conjunto de artefatos gerados durante a atividade de testes (planos de teste, scripts, entre outros).

2.6 Considerações Finais

Por ser uma disciplina considerada recente, a atividade de testes possui termos que muitas vezes são novos e, portanto, faz-se necessária a revisão desses conceitos, objetivando preparar o leitor para o bom entendimento das seções seguintes. Este capítulo foi dedicado à uma revisão dos principais conceitos relacionados a testes.

O ponto mais importante a ser observado neste capítulo são as atividades mostradas no processo de teste, pois o foco deste trabalho está em como melhorar essas atividades, a fim de trazer resultados significativos para o processo como um todo e entender que a atividade de teste precisa ser vista como sendo fundamental no desenvolvimento de software e que necessita ser realizada de forma sistemática e estruturada, visando garantir a qualidade dos sistemas.

O próximo capítulo se dedica ao detalhamento da automação e de sua importância no processo de testes, como forma de tornar mais eficiente a atividade de execução.

3 AUTOMAÇÃO DE TESTES

Este capítulo tem como objetivo expor os principais conceitos da automação de testes. Na seção 3.1 é apresentada uma visão geral do contexto da automação. Na seção 3.2 os fundamentos da automação são levantados. Na seção 3.3 são mostradas as principais técnicas para geração de scripts. Na seção 3.4 duas ferramentas são apresentadas. Na seção 3.5 o problema de reconhecimento de objetos dinâmicos é investigado. Por fim, na seção 3.6 são apresentadas as considerações finais.

3.1 Visão Geral

Como dito no capítulo anterior, processos de testes são importantes, para que a entrega do produto final tenha requisitos de qualidade atendidos de uma forma mais satisfatória. Sabendo da necessidade do mercado por qualidade, os processos desenvolvidos são postos em prática e de acordo com o nível de maturidade de cada organização, uma busca pela melhoria das atividades de testes é uma constante.

Baseado nos dados coletados ao fim de cada projeto, os processos passam por uma avaliação e pontos de melhoria são propostos para serem implementados em iterações futuras. Como a atividade de teste tem a maior parte do seu esforço concentrado entre as fases de arquitetura (análise e modelagem) e execução, há um interesse maior em tornar estas etapas mais produtivas.

Este capítulo mostra como a automação de teste é fundamental para agilizar a etapa de execução, apresentando os fundamentos do processo, as diferentes técnicas de geração de *scripts*, vantagens de sua utilização, principais ferramentas e suas limitações.

3.2 Fundamentos da Automação

A engenharia de testes, assim como qualquer outra engenharia, define a criação de produtos que sejam de qualidade, utilizando processos eficientes e, acima de tudo, economicamente viáveis.

Testar softwares é uma atividade estratégica, visto que não há recursos de tempo e orçamento ilimitados, ou seja, o arquiteto de teste, de posse de documentos de especificação de requisitos, deve criar cenários de tal forma que este conjunto reduzido possa cobrir o máximo de situações a serem validadas, utilizando o mínimo de recursos possível.

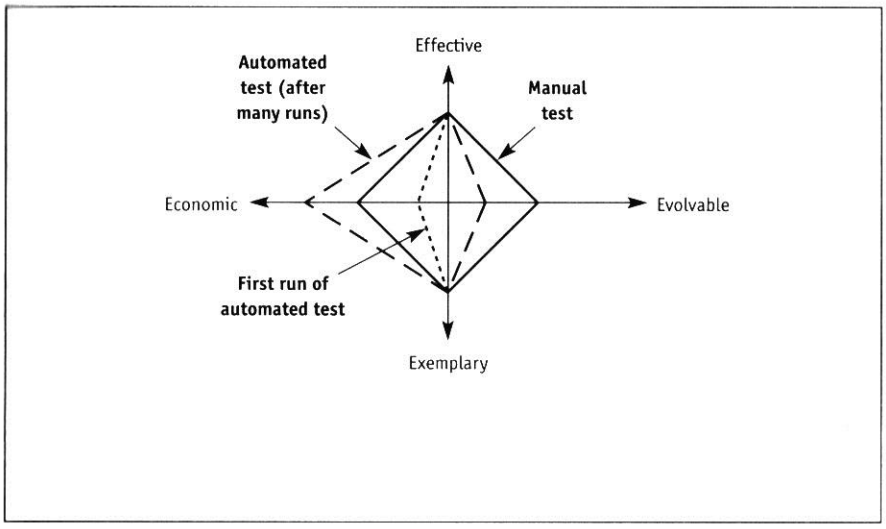
Segundo (3), um caso de teste deve ser exemplar, efetivo, fácil de evoluir e econômico. Um cenário de teste para ser exemplar deve testar diferentes questões ao mesmo tempo, com o objetivo de reduzir a quantidade de testes a ser criada. Efetivo no sentido de realmente ser capaz de detectar problemas. Econômico em relação à sua análise, execução e depuração, e, finalmente, fácil de evoluir caso o sistema sofra alterações.

Fazer com que os casos de testes atinjam essas quatro características não é uma tarefa trivial e por isso técnicas para auxiliar o processo foram desenvolvidas. A automação emergiu dessa necessidade e hoje é largamente usada pelas fábricas de teste e software.

A decisão de automatizar testes não pode ser baseada somente no desejo de economizar recursos na fase de execução, pois o processo de automação num primeiro momento é mais custoso do que o manual, o que acarreta frustração quando da tentativa de sua implantação. A automação é válida quando seu uso impacta de forma positiva em algum dos atributos de um bom caso de teste, ou seja, não adianta ter um teste automático e outro manual que tenham o mesmo nível de efetividade, a mesma freqüência de execução, pois se terá apenas o mesmo teste que executa mais rápido a um custo bem maior. (3)

A figura abaixo, extraída de (3), mostra a relação entre testes automáticos e manuais, do ponto de vista dos atributos definidos para um caso de teste de qualidade. Uma análise da imagem evidencia que testes automáticos só se tornam viáveis após vários ciclos de execução e mostra também que testes manuais são mais fáceis de manter, caso haja mudanças no sistema.

Figura 1 - Comparação entre as abordagens automática e manual em relação aos quatro atributos de um caso de teste



Fonte: FEWSTER e GRAHAM, pag. 5

Além dos papéis já conhecidos para a atividade de teste (arquiteto, líder e testador), a automação trás mais um papel que seria o do automatizador. Responsável por criar os *scripts* de testes, o automatizador não precisa possuir conhecimento em testes, bastando ter habilidades em codificação e dominar a ferramenta a ser utilizada como apoio à atividade. (3)

Ferramentas de apoio às atividades de desenvolvimento de software têm sido criadas e seu uso bem aceito no mercado. O processo de teste também não fugiria à regra e a automação só foi possível devido à disponibilidade de ferramentas para este fim.

A automação possui pontos favoráveis e negativos quanto ao seu uso, o que exige planejamento e negociações prévias à sua implantação. As vantagens mais evidentes seriam a possibilidade de re-executar testes (testes de regressão), após uma alteração no sistema, mais rapidamente; a facilidade em executar um volume maior de cenários num menor tempo; capturar eventos difíceis de detecção pelo método manual; aproveitamento de *scripts* gerados para outros cenários. As principais desvantagens são manutenção dos *scripts*, que podem precisar ser atualizados ou recriados devido a mudanças de sistema; falha nas ferramentas utilizadas no processo; estrutura organizacional insuficiente para a plena prática da atividade. (3)

A seguir são apresentadas as principais técnicas para criação de *scripts* de testes.

3.3 Técnicas para Geração de Scripts

O principal artefato gerado com a atividade de automação é o *script* de teste, que pode ser criado, aplicando-se diversas técnicas. Dependendo dos objetivos do teste e do objeto de teste, deve-se optar por uma ou outra abordagem ou mesmo uma combinação entre elas.

As técnicas desenvolvidas para criação de *scripts* são a *Record and playback, programação de scripts, orientado a dados* e *orientado à palavra-chave*. (8)

Na abordagem orientada a dados, as informações que precisam estar inseridas nos testes são colocadas em arquivos separados para tornar o *script* independente dos dados e assim utilizá-los para realizar outros testes com arquivos de dados diferentes. Já na orientada a palavras-chave, os *scripts* gerados são mapeados em comandos como se fossem funções, podendo ser utilizadas em diferentes testes. (9)

A composição dos *scripts* proposta neste trabalho está baseado no uso de ferramentas que implementam as técnicas *Record and Playback* e a programação de *scripts*, que são descritas abaixo:

Record and playback

Utilizado na abordagem funcional (caixa-preta), no qual os testes precisam interagir com a interface do sistema, a técnica *Record and playback* consiste na utilização de ferramentas que gravam as ações no sistema para realizar determinada funcionalidade. Essas gravações são transformadas em *scripts* prontos para executar e então simular o caso de teste gravado.

Esta técnica tem a vantagem de ser simples e de fácil aplicação, necessitando dominar apenas a ferramenta a ser utilizada. No entanto, seus problemas são mais extensos, pois a gravação realizada é valida apenas para uma versão do sistema e caso haja alterações, novas gravações deverão ser feitas, evidenciando o alto custo na manutenção desses *scripts*. Outro problema é a baixa taxa de reuso, visto que é bastante difícil desmembrar um *script* para a formação de outros testes. (8)

Programação de Scripts

Técnica utilizada em associação com a *Record and playback* e que permite que os scritps gravados possam ser alterados de forma que o automatizador possa codificar detalhes que não são capturados numa gravação comum, tornando o caso de teste mais eficaz. Devido à possibilidade de edição dos *scripts*, a taxa de reuso desta técnica se mostra bem melhor que a anterior. (8)

Essas duas últimas abordagens são implementadas pelas ferramentas utilizadas para executar os *scripts* gerados pelo produto tratado neste trabalho. A seguir um detalhamento delas é feito.

3.4 Ferramentas para Automação

Diversas ferramentas têm sido desenvolvidas para dar suporte ao processo de teste de uma forma global, onde já estão em uso ferramentas para geração de casos de

teste, de análise de cobertura, de análise estática e dinâmica do código, entre outras. Esta seção trata apenas das ferramentas utilizadas para a geração de *scripts* de testes automáticos.

Como visto anteriormente, as diferentes técnicas de geração de *scripts* possuem características específicas que, para serem implementadas, precisam de ferramentas apropriadas para realizar aquela atividade. As ferramentas mais comuns estão inseridas na abordagem *Record and playback*, mas, atualmente, na maioria delas, é possível implementar as outras abordagens usando a mesma ferramenta.

A decisão de escolher determinado sistema para geração dos *scripts* precisa estar fortemente baseada nos objetivos da empresa, no tamanho da equipe e no tipo de teste e abordagem a ser utilizada. (3)

Uma lista com ferramentas que atendem às diversas classificações de testes seria bastante extensa. A seguir são descritas as ferramentas usadas na solução do problema proposto no trabalho.

3.4.1 Rational Functional Tester (RFT)

Ferramenta que compõe o grupo de pacotes de apoio ao desenvolvimento de software da IBM e que se destina à criação de *scripts* de testes baseados na técnica de gravação e execução. O aplicativo também permite que os *scripts* gerados possam ser editados de acordo com as necessidades do testador. A possibilidade de separar os dados dos scritps gerados também é uma outra facilidade.

O RFT dá, ainda, suporte à modularização ou reuso dos scritps, como definido na técnica orientada à palavra-chave, mostrando que uma mesma ferramenta pode estar aplicada a diferentes abordagens. (10)

3.4.2 Selenium

Conjunto de ferramentas destinadas à realização de testes funcionais em aplicações web, gerando *scripts* através da gravação das ações no sistema.

O *Selenium* é composto de três módulos, o *Selenium* IDE (Integrated Development Environment), o *Selenium* RC (*Remote Control)* e o *Selenium* Grid. Essas ferramentas funcionam em conjunto, no qual o papel do *Selenium* IDE, complemento do navegador Firefox, é guiar o usuário na gravação da interação com a página *web*. O RC permite que os *scripts* gravados no IDE, após exportados, sejam editados, utilizando-se diversas linguagens de programação. O Grid é responsável por tornar possível a execução de vários testes em paralelo. (11)

A próxima seção descreve um problema comum que ocorre com o uso de ferramentas de geração de *scripts* por meio da técnica de gravação. Entender esse problema é importante para contextualizar as limitações ocorridas na avaliação do produto.

3.5 Automação e objetos dinâmicos

Apesar do uso de ferramentas de automação estar bastante disseminado no mercado, tanto a técnica quanto os sistemas possuem limitações à sua aplicação.

Como mostrado anteriormente, automatizar testes não é uma tarefa trivial, exige planejamento, organização e, além disso, nem todas as situações são possíveis de serem automatizadas. Ultrapassando esta questão e assumindo que os cenários de teste possam ser automatizados, a escolha da ferramenta aparece como uma segunda restrição do método, pois é necessário garantir que a mesma seja capaz de interagir corretamente com o sistema sob teste.

As ferramentas disponíveis no mercado, sobretudo as citadas neste capítulo, funcionam satisfatoriamente quando os sistemas submetidos a elas estão codificados utilizando tecnologia compatível com a especificada nas ferramentas de automação.

Devido à grande oferta de *frameworks* para desenvolvimento *web*, os componentes de interface das aplicações são inseridos no código de forma dinâmica e as ferramentas de automação possuem dificuldade de detecção desses elementos devido à mudança dos atributos no momento da execução do *script* gravado.

Reconhecimento de objetos dinâmicos é um dos problemas que a automação de teste precisa lidar. Para um melhor entendimento do problema, suponha que durante a gravação de uma funcionalidade, a aplicação gere, por meio de *Javascript,* uma lista ou uma árvore contendo resultados de uma busca. Essa lista ou árvore de resultados é composta por estruturas geradas dinamicamente e inseridas no código HTML. No momento da gravação, esses objetos são atribuídos a um identificador e o *script* é gerado. Quando são postos em execução, esses objetos são gerados novamente e ganham novos identificadores que divergem daquele presente no *script* de gravação. O resultado deste fato é uma falha no teste, pois os elementos com os quais o sistema automatizador precisa interagir não estão “visíveis”.

Sistemas de automação mais robustos possuem a implementação de heurísticas de busca de objetos, ou seja, tais ferramentas catalogam um conjunto de informações do objeto detectado no momento da gravação (mapa de objetos). Atributos como nome, identificador, posição entre outros são utilizados para a realização do reconhecimento do objeto. Outra alternativa, para garantir a detecção, é o automatizador ter acesso ao código do sistema em teste, identificar a forma como os objetos são criados e interceptá-los via código. Esta última alternativa não é muito eficiente, visto que é necessário o acesso ao código, para entender e programar o *script* adequadamente.

Para visualizar a ocorrência desse problema, usou-se o *TestLink*, ferramenta de gerenciamento de artefatos de teste (casos de testes, suítes e planos de teste), como aplicação *web* a ser testada. Foi criado um cenário de teste e a gravação realizada com o uso do *Selenium*. A figura abaixo mostra o *script* gerado:

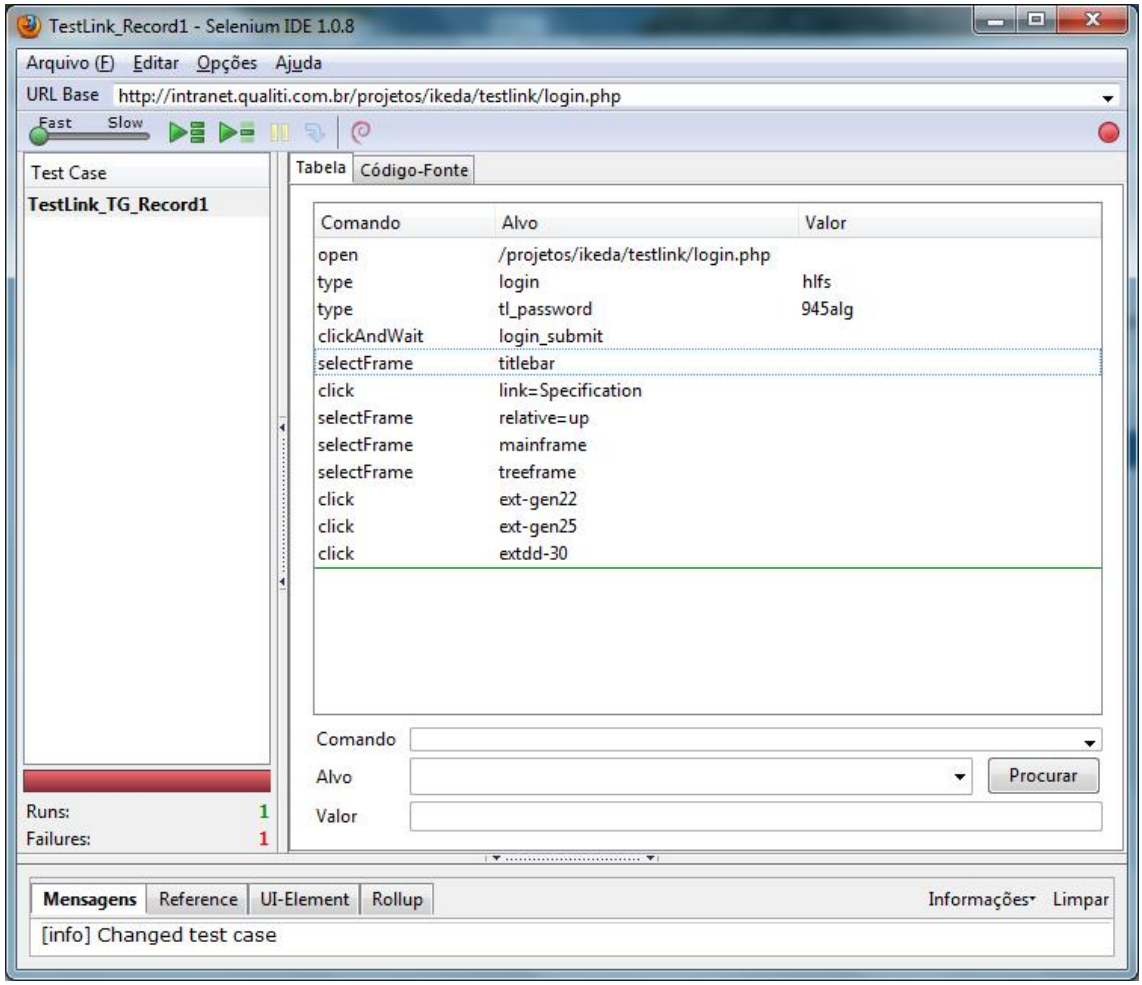


Figura 2 - Gravação de um cenário de teste simples que contém objetos dinâmicos

O *script* resultante foi posto em execução novamente e uma falha ocorreu como evidencia a figura 3:

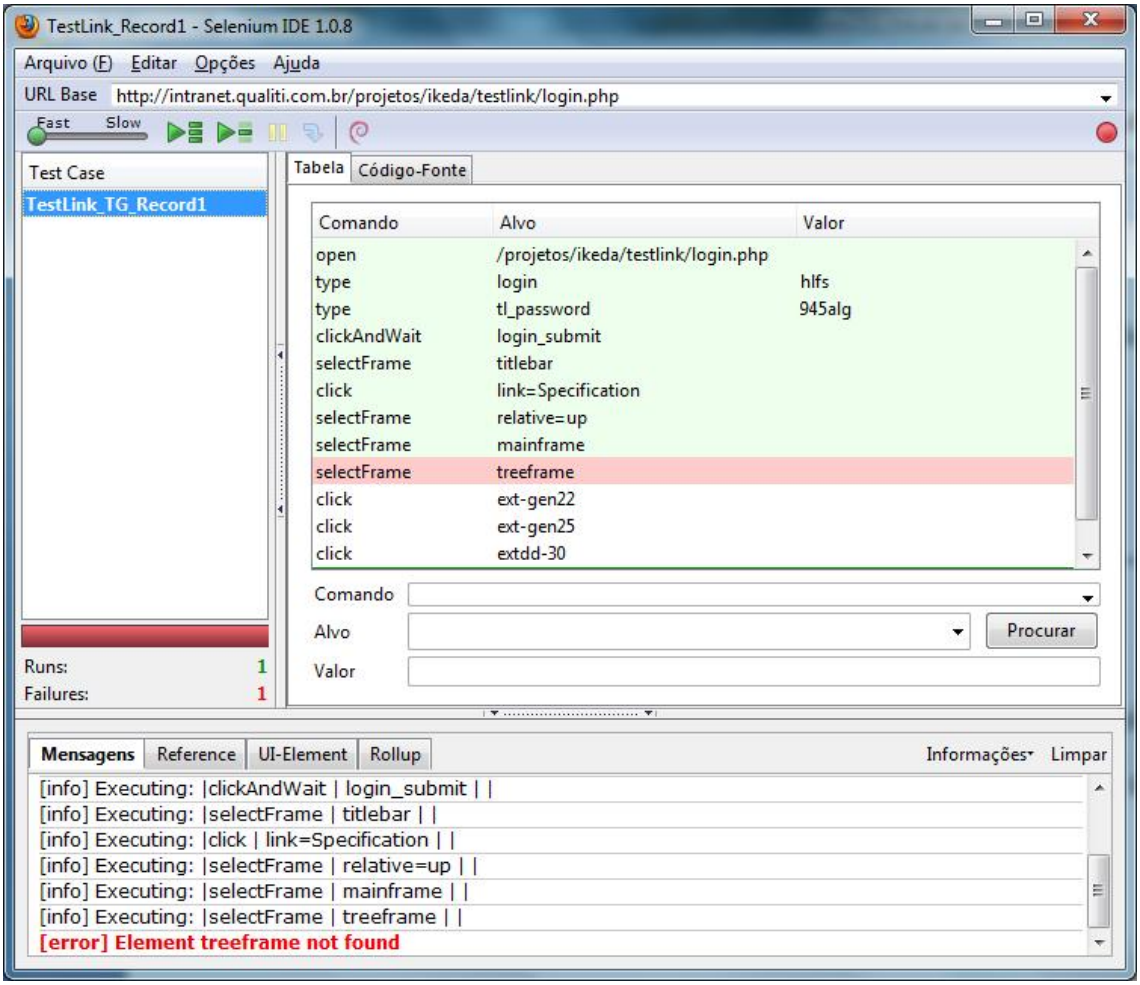


Figura 3 - Evidência da falha do teste por não reconhecer um objeto dinâmico

Nesta execução a árvore que lista os casos de teste no *TestLink* não foi reconhecida, causando uma falha no teste.

Para confirmar a geração dinâmica do objeto em questão, o mesmo cenário foi gravado novamente e o *script* gerado é descrito abaixo:

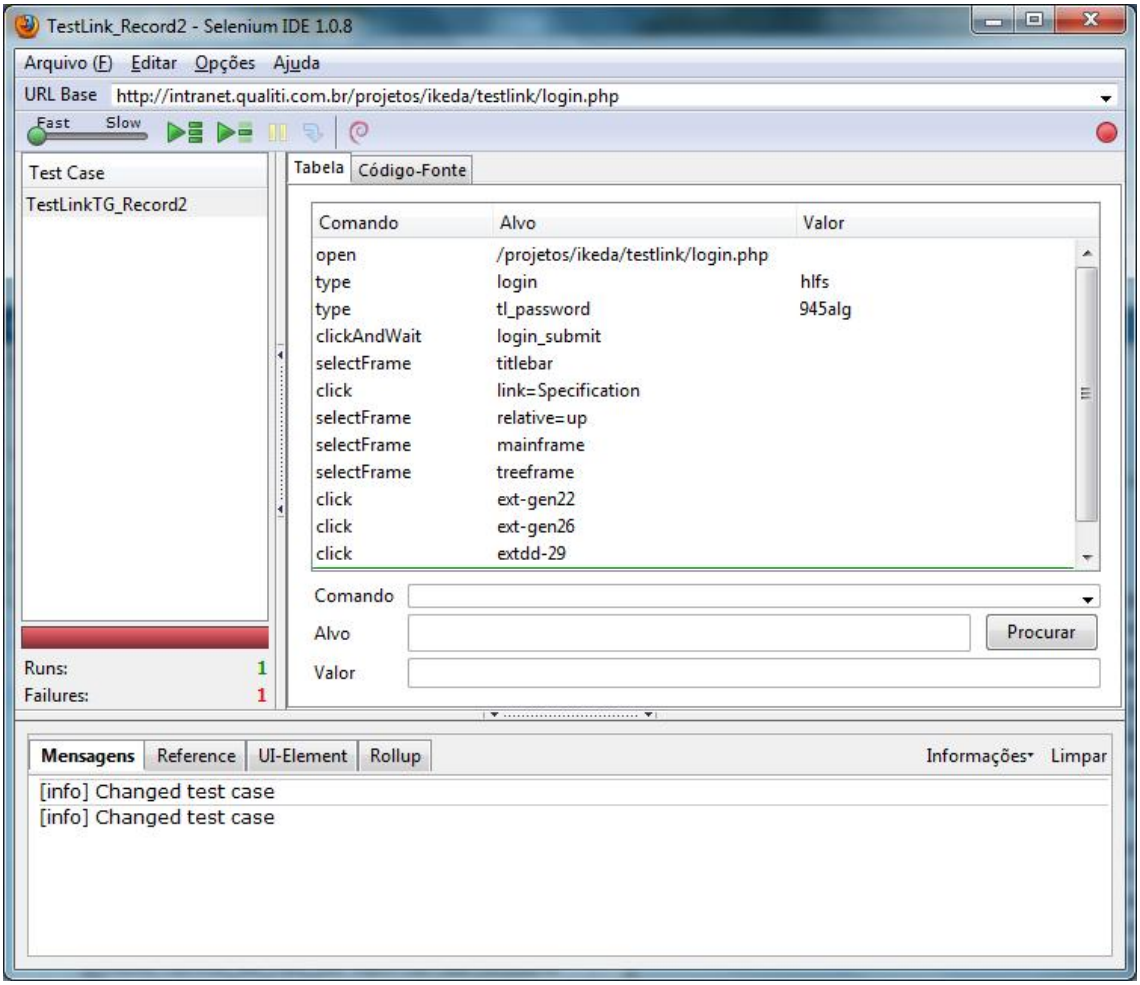


Figura 4 - Mesmo cenário gravado novamente. Observe que os identificadores dos objetos mudaram

A figura acima mostra que, numa segunda gravação, os atributos identificadores de alguns objetos sofreram alterações. Apesar do elemento *treename* não ter tido seu nome alterado, a busca no *Selenium* se deu utilizando outro atributo que mudou de valor, causando a falha de reconhecimento.

Embora seja custosa em fases iniciais e precise lidar com problemas como o citado acima, a automação de testes sem dúvida é uma técnica bastante útil nos cenários

indicados para sua implantação e traz benefícios palpáveis para as empresas que a utiliza.

3.6 Considerações Finais

Este capítulo tratou da automação de teste, técnica muito útil para tornar o processo de testes mais ágil, mas que necessita de maturidade para sua implantação ter sucesso. Foram mostradas as ferramentas utilizadas neste trabalho para gerar os *scripts*, além das vantagens e limitações da automação.

Outro ponto importante a ser notado é que técnicas para melhorar o processo de testes estão sempre sendo pesquisadas e desenvolvidas, com o intuito de tornar a aplicabilidade da disciplina mais fácil e eficaz.

Entender os conceitos de automação, vantagens e suas limitações é necessário, pois a solução do problema relatado neste trabalho passa por uma adaptação da técnica *Record and Playback* para prover o reuso dos *scripts* gerados.

O próximo capítulo faz uma revisão dos conceitos da tecnologia utilizada para o desenvolvimento da aplicação e introduz a TaRGeT, ferramenta que terá essa funcionalidade integrada em sua linha de produtos.

4 DESENVOLVIMENTO DE PLUG-INS E A TARGET

Este capítulo tem como objetivo elucidar questões relativas ao desenvolvimento de plug-ins, explicando os principais conceitos e tecnologias utilizadas. Na seção 4.1 uma visão geral do capítulo é dada. Na seção 4.2, os principais elementos do desenvolvimento orientado a plug-in são abordados. Na seção 4.3, a TaRGeT é introduzida, sua arquitetura e principais funcionalidades mostradas. Por fim, na seção 4.3 são apresentadas as considerações finais.

4.1 Visão Geral

O uso da TaRGeT como ferramenta para geração de cenários de teste direcionou a construção da solução através de um *plug-in*, para que o mesmo pudesse ser facilmente integrado a sua arquitetura, que está fundamentada no desenvolvimento orientado a *plug-in.*

O conceito de *plug-in* está embasado na idéia de um sistema central poder receber contribuições por meio de componentes que foram implementados independentemente e que são facilmente integrados à aplicação principal. Tais estruturas possuem formas específicas para seu desenvolvimento e serão objetos de estudo neste capítulo.

Entender os principais conceitos e a forma como a TaRGeT é utilizada, é essencial para perceber o papel do produto desenvolvido no contexto da ferramenta.

4.2 Desenvolvimento de Plug-ins – Fundamentos

Uma explicação do papel e funcionamento de *plug-ins* pode ser realizada fazendo-se uma analogia com a arquitetura orientada a serviços (*Service-oriented Architecture – SOA*), na qual vários serviços são criados e publicados, para que sejam consumidos por outros componentes ou associados a outros serviços e disponibilizar novas funções para as entidades que os solicitam.

Trazendo para o contexto de *plug-ins*, o parágrafo anterior diz que um *plug-in* nada mais é do que um componente que oferece um conjunto de serviços e que pode ser agregado a outros, objetivando oferecer, ao componente central, novas funcionalidades.

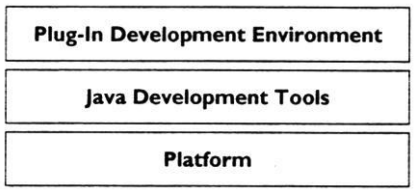
Segundo (12), um *plug-in* é, em termos técnicos, um arquivo JAVA (JAR) que define uma estrutura auto-contida e auto-descrita. Auto-contida por que tudo o que é necessário para sua execução está dentro do *plug-in* e auto-descrita, pois deve ser clara ao informar a que se destina e o que necessita de entidades externas.

O fluxo de atividades para a criação de *plug-ins* obedece ao que foi estipulado pelo PDE (*Plug-in Development Environment)*, ambiente integrado ao Eclipse e que possui conceitos e atividades que devem ser seguidas.

O Eclipse é uma ferramenta de desenvolvimento construída através da integração de vários *plug-ins* e, portanto, oferece, como definido no PDE, pontos de extensão e extensões que o compõem. Pontos de extensão são partes que o eclipse publica como sendo possíveis de receber implementações e contribuições. As extensões, por sua vez, são os componentes criados para extender o ponto de extensão (12).

A figura abaixo ilustra a arquitetura na qual o eclipse está construído:

Figura 5 - Arquitetura Eclipse



Fonte: GAMMA e BECK, pag. 5

A plataforma representa a infra-estrutura comum independente da linguagem de programação. As ferramentas de desenvolvimento Java (*Java Development Tools – JDT*) tornam o eclipse uma IDE para programação Java e o PDE é uma especialização do JDT para possibilitar o desenvolvimento de *plug-ins.* (13)

Dentre os componentes comuns da plataforma eclipse, trazidos por Gamma e Beck (13), merecem destaque os seguintes elementos:

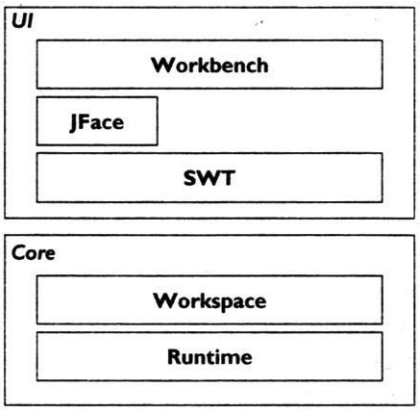
Standard Widget Toolkit (SWT): Ferramenta que fornece um conjunto de componentes gráficos padrões para desenvolvimento.

JFace: Um conjunto menor de componentes de interface, construído com base no SWT e que dá suporte às tarefas de interface do usuário.

Workbench: Elemento gráfico principal do paradigma de interface do Eclipse. É na *workbench* que outros componentes como editores, *views* e perspectivas são construídos.

As figuras 6 e 7 exibem a forma como a plataforma eclipse está organizada e mostra os componentes que podem estar presentes na *Workbench*, respectivamente:

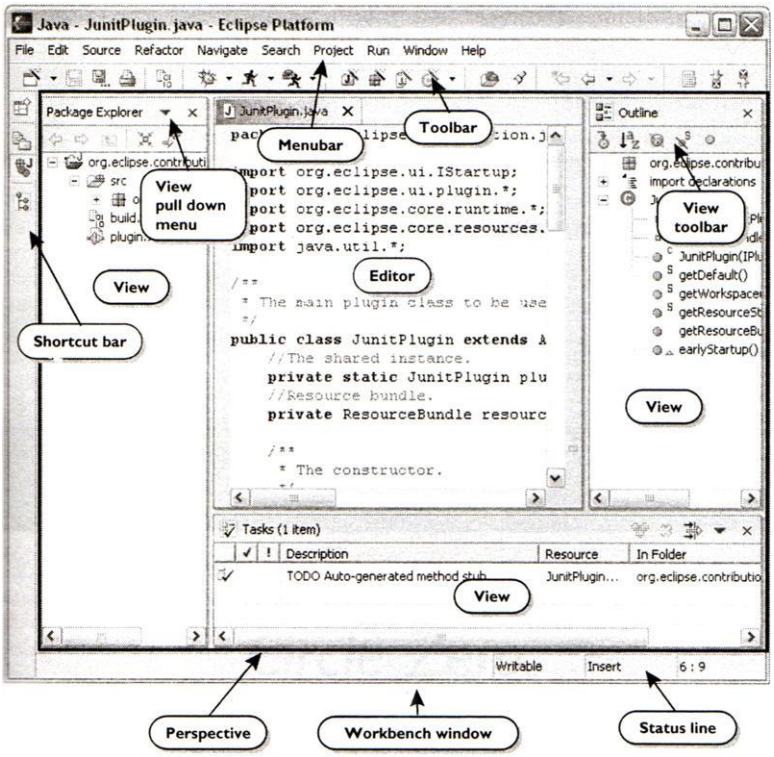
Figura 6 - Estrutura da plataforma Eclipse



Fonte: GAMMA e BECK, pag. 6

A figura acima ilustra a forma como os componentes da plataforma Eclipse estão organizados. Os elementos detalhados anteriormente compõem a maneira que itens de interface com o usuário são criados.

Abaixo temos a *workbench* do Eclipse, na qual os elementos mostrados pelas setas são criados, utilizando elementos gráficos presentes no *SWT* , *JFace* e no *RCP (Rich Client Platform)*.



Fonte: GAMMA e BECK, pag. 7

O RCP é uma plataforma que oferece os componentes necessários para a criação de aplicações baseadas no Eclipse, onde os itens de interface podem ser estendidos, ou seja, para criar editores, perspectivas, contribuir no *menu bar*, deve-se utilizar elementos do RCP e extender as funcionalidades da ferramenta.

Após essa visão geral dos termos e componentes presentes no desenvolvimento orientado a *plug-in,* a TaRGeT pode ser estudada de forma detalhada, com o objetivo de entender sua arquitetura e principais funcionalidades.

Como visto anteriormente, o processo de testes está sempre dispondo de novas aplicações que auxiliem em alguma atividade e a torne mais ágil. A próxima seção esclarece como a etapa de análise e modelagem (arquitetura de testes) é otimizada com o uso da TaRGeT.

4.3 TaRGeT – Introdução

A TaRGeT *(Test and Requirements Generation Tool)* tem como objetivo a criação de cenários de teste que estão diretamente ligados ao documento de casos de uso do sistema. Esta ferramenta é derivada da técnica de automação de testes baseada em modelos *(Based-Model Testing)*, na qual os casos de testes são gerados a partir da definição de um modelo formal do sistema a ser testado. (4)

Com o intuito de diminuir o esforço na criação de cenários de teste com base nos requisitos, foi definido para a TaRGeT um modelo para descrever os casos de uso do sistema. No modelo existem campos referentes às ações do usuário, resposta do sistema e pré-condições para cada passo do caso de uso. É possível também a criação de fluxos alternativos a partir do reuso de passos já mapeados.

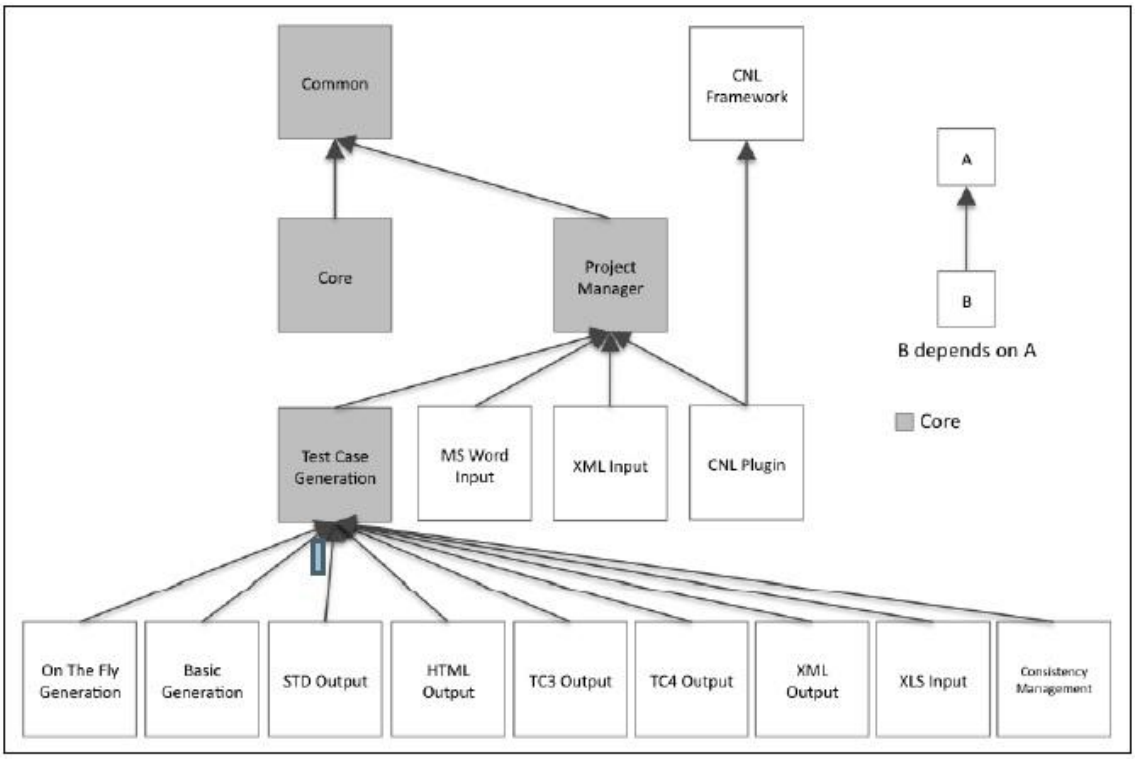
Com o uso da TaRGeT é possível criar casos de teste que são combinações dos passos descritos no modelo formal de caso de uso aceito pela ferramenta. Esses testes podem ser gerenciados pela interface do sistema e suítes podem ser criadas com os testes selecionados com base nos critérios de cobertura, requisitos exercitados entre outros filtros.

A seguir os principais componentes que formam o sistema são descritos.

Organização dos Componentes

Desenvolvida por meio da plataforma RCP, a TaRGeT é formada por *plug-ins* que possuem responsabilidades específicas. A figura abaixo, extraída de (4), mostra como a arquitetura e as dependências dos *plug-ins* estão organizadas.

Figura 8 - Arquitetura TaRGeT



Fonte: FERREIRA et al, pag. 5

Para entender melhor a figura, é necessário esclarecer que a TaRGeT compõe uma linha de produtos, ou seja, várias *features* ou *plug-ins* foram desenvolvidos para clientes específicos, compondo produtos específicos que atendem a diferentes necessidades. Os *plug-ins* exibidos em cinza correspondem ao núcleo comum a toda a linha de produtos, isto é, são os responsáveis pelo funcionamento básico da ferramenta. Os componentes mostrados em branco ou sem preenchimento são funcionalidades adicionadas para suprir determinado fim, como, por exemplo, exportar uma suíte de testes num determinado formato ou em outro.

Abaixo os principais módulos, importantes para a construção do produto proposto, são descritos, segundo a visão de (4):

TaRGeT Test Case Generation: realiza a geração de suítes de teste que podem ser exportadas em diferentes formatos, além de possibilitar a extensão dessa funcionalidade para implementar novos formatos de saída.

Dentre os *plug-ins* que não fazem parte do núcleo básico da TaRGeT, uma atenção ao *On The Fly Generation* deve ser demandada.

TaRGeT On The Fly Generation: *plug-in* responsável por gerenciar os casos de testes gerados pelo *TaRGeT Test Case Generation*. Esse gerenciamento consiste em criar suítes aplicando filtros de seleção e exportando para vários formatos como Excel e XML. Como o gerenciamento dos *scripts* e seu reuso dependem dos testes gerados, a solução desenvolvida estende as funcionalidades presentes no *On The Fly Generation*.

4.4 Considerações Finais

O presente capítulo se dedicou a apresentar os conceitos básicos da criação de *plug-ins*, mostrando que os pacotes disponibilizados são bastante úteis e poderosos para a criação de aplicações como a TaRGeT.

Sabendo que as etapas de arquitetura e execução são as que demandam mais tempo no processo de teste e que a TaRGeT melhora o tempo de criação dos casos de teste, fazer com que a ferramenta seja útil para a automação é um ponto de melhoria para torná-la mais completa.

O próximo capítulo se dedica aos detalhes do componente desenvolvido, mostrando os recursos utilizados, as funcionalidades disponíveis e o fluxo de trabalho para a correta aplicação do *plug-in.*

5 PLUG-IN PARA GERAÇÃO DE SCRIPTS

Este capítulo visa apresentar o plug-in TaRGeT Automatic Scripts Generation criado para introduzir o reuso de scripts de testes automáticos na TaRGeT. Na seção 5.1 uma visão geral do capítulo é dada. Na seção 5.2, um levantamento do fluxo de trabalho é descrito, para o entendimento do funcionamento do componente. Na seção 5.3 as funcionalidades adicionadas pelo plug-in são detalhadas. A seção 5.4, traz uma análise funcional do RFT e Selenium. A seção 5.5 mostra o fluxo de uso do plug-in. Por fim, na seção 5.6 são apresentadas as considerações finais.

5.1 Visão Geral

Os capítulos anteriores foram importantes para mostrar os conceitos de teste de software e seus processos, assim como a importância da automação de testes para reduzir o tempo de execução das suítes. Nesse contexto, foi apresentada a TaRGeT, ferramenta que se destina a apoiar o processo de modelagem dos testes, e a possibilidade de extender funcionalidades através da programação baseada em *plug-in.*

O presente capítulo tem como objetivo mostrar as etapas realizadas, com o fim de entregar um *plug-in* capaz de reutilizar *scripts* de testes gravados em ferramentas de testes funcionais. A próxima seção detalha o fluxo de desenvolvimento realizado.

5.2 Metodologia de Desenvolvimento

A experiência da fábrica de teste tratada neste trabalho evidencia que é bastante comum, em seus projetos, que casos de testes possuam em sua estrutura pontos semelhantes para serem executados. Ferramentas de gravação de testes geram os *scripts* após o cenário ter sido gravado por completo, ou seja, aqueles pontos semelhantes são regravados sempre que aparecem. Ao mesmo tempo, "quebrar" esse *script* para derivar novos fluxos não é um trabalho muito eficiente para ser realizado pelo automatizador.

O primeiro ponto trabalhado foi encontrar uma forma de fragmentar esses *scripts* para posteriormente remontá-los automaticamente. Para esta etapa, as ferramentas de automação *Selenium* e RFT foram analisadas e se observou que retirar os passos que compunham esses *scripts* era bastante custoso, no caso do *Selenium*, devido ao processo de identificar onde um passo acaba e o outro começa. No caso do RFT a dificuldade é ainda maior, pois os *scripts* gerados estão associados a arquivos de mapas de objetos capturados durante a gravação. Em outros termos, para particionar os *scripts* do RFT, era necessário fazer o mesmo com o mapa de objetos e a alteração desses arquivos afeta o pleno funcionamento da ferramenta.

Diante dos fatos relatados acima, uma outra abordagem para tratar os *scripts* gravados foi concebida. A seguir os passos da metodologia adotada:

1. Criar casos de uso no modelo formal de documento aceito pela TaRGeT.
2. Gerar os testes usando o *On The Fly Generation*.
3. Gravar os passos dos casos de teste.
4. Juntar manualmente esses *scripts* num mesmo arquivo, simulando a montagem do *script* do teste completo.
5. Executar o *script* resultante na ferramenta na qual os passos foram gravados.

Essa abordagem foi possível em ambas as ferramentas (*Selenium* e RFT). Os *scripts* gerados pelo *Selenium*, quando exportados para Java, podem ser postos na seqüência correta de execução, após um tratamento não muito complexo dos *scripts* dos passos. Com relação ao RFT, a viabilidade da correta execução desta metodologia foi confirmada devido à possibilidade de modularizar os casos de testes, por meio da chamada de *scripts* mapeados como funções.

Para exemplificar o que foi exposto, suponha que um caso de teste possua dois passos. Esses passos são gravados e os *scripts* resultantes são unidos para implementar o *script* do teste completo. As figuras 9, 10 e 11 mostram como seria essa construção no Selenium:

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;

	public class 4185#UC_01#1M extends SeleneseTestCase {
	   public void setUp() throws Exception {
	         setUp("http://www.ibm.com/", "*chrome");
	   }
	   public void test4185#UC_01#1M() throws Exception {
	         selenium.open("/us/en/sandbox/ver2/");
	         assertEquals("IBM - United States", selenium.getTitle());
	         selenium.click("//img[@alt='How could your city grow smarter? Find answers in the Smarter Cities Challenge.']");
	         selenium.waitForPageToLoad("30000");
	   }
	}
```

Figura 9 - Script gerado para o passo 1 do caso de teste hipotético

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;

public class 4185#UC_01#2M extends SeleneseTestCase {
	public void setUp() throws Exception {
		setUp("https://smartercitieschallenge.org/", "*chrome");
	}
	public void test4185#UC_01#2M() throws Exception {
		selenium.open("/?link=ibmhpcs2/corp/smarterplanet/cities_challenge");
		assertEquals("IBM Smarter Cities Challenge: Overview", selenium.getTitle());
		selenium.click("//img[@alt='About the Challenge']");
		selenium.waitForPageToLoad("30000");
		assertEquals("IBM Smarter Cities Challenge: About the Challenge", selenium.getTitle());
	}
}
```

Figura 10 - Script gerado para o passo 2 do caso de teste hipotético.

A figura abaixo mostra, circundado em vermelho, o trecho de código referente ao passo 1 do caso de teste e em verde o trecho relativo ao passo 2. Dessa forma o teste completo do cenário está pronto para executar, pois o método *setUp()*, que inicia a aplicação, foi extraído do passo 1 e os blocos de código do primeiro e segundo passos foram postos na ordem apropriada.

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;

public class completeTestCase extends SeleneseTestCase {
	public void setUp() throws Exception {
		setUp("http://www.ibm.com/", "*chrome");
	}
	public void completeTestCase() throws Exception {
		selenium.open("/us/en/sandbox/ver2/");
		assertEquals("IBM - United States", selenium.getTitle());
		selenium.click("//img[@alt='How could your city grow smarter? Find answers in the Smarter Cities Challenge.']");
		selenium.waitForPageToLoad("30000");

		assertEquals("IBM Smarter Cities Challenge: Overview", selenium.getTitle());
		selenium.click("//img[@alt='About the Challenge']");
		selenium.waitForPageToLoad("30000");
		assertEquals("IBM Smarter Cities Challenge: About the Challenge", selenium.getTitle());
	}
}
```

Figura 11 - Composição dos scripts dos passos para formar o script do caso de teste

As figuras 12, 13 e 14 representam a mesma metodologia, agora aplicada ao RFT.

Figura 12 - Script do passo 1 de um caso de teste no RFT

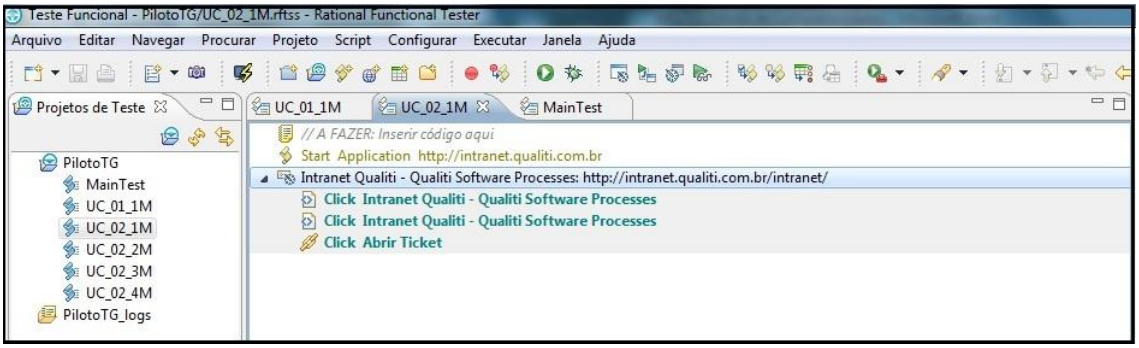


Figura 13 - Script do passo 2 de um caso de teste no RFT

Após a gravação isolada dos passos do caso de teste no RFT, os mesmos podem ser combinados através da função *callscript* disponibilizada pela *API* da ferramenta. A figura abaixo mostra como o teste completo deve estar escrito.

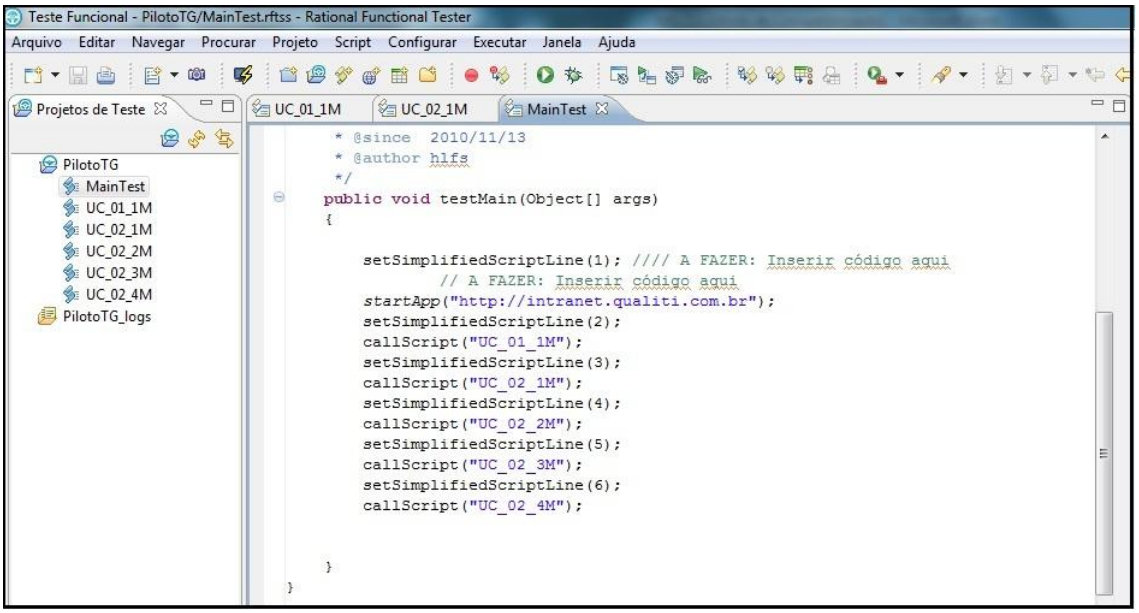


Figura 14 - Caso de teste montado a partir da chamada dos scripts dos passos

Após aplicar a metodologia usando ambas as ferramentas, verificou-se que o *plug-in* deveria ser capaz de fazer essa montagem automaticamente, recebendo como informação os arquivos dos *scripts* dos passos e dando como resposta arquivos representando os testes automatizados.

A figura abaixo ilustra o macro fluxo de trabalho proposto para o correto uso da TaRGeT, após a integração com o *plug-in* de geração de *scripts*.

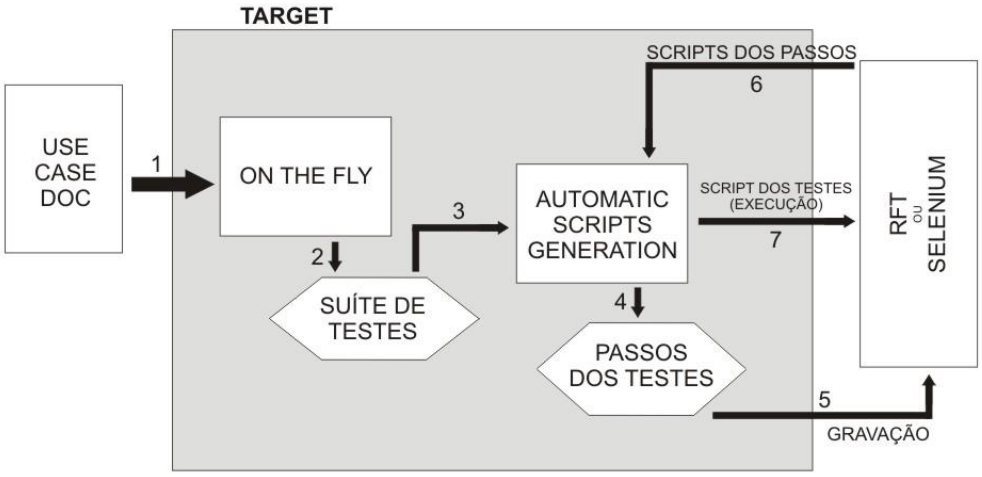


Figura 15 – Macro fluxo para geração de scripts na TaRGeT

A etapa 1 refere-se à construção do documento de caso de uso do sistema a ser testado e importação do arquivo na TaRGeT, que utiliza o *On The Fly Generation* para gerar a suíte de teste (etapa 2). Essa suíte é analisada pelo *Automatic Scripts Generation* (etapa 3), que lista todos os passos que aparecem no conjunto de testes (etapa 4). Cada passo é, então, gravado utilizando uma das ferramentas disponíveis (etapa 5) e os *scripts* gerados são importados novamente para a TaRGeT (etapa 6), para que o *plug-in* de geração de *scripts* possa agrupá-los e gerar os casos de testes automáticos que estão prontos para execução (etapa 7).

A seção seguinte lista as funcionalidades criadas que visam implementar os cenários aqui descritos.

5.3 Funcionalidades Adicionadas

O *plug-in* desenvolvido está baseado nos seguintes requisitos:

1. O sistema deve ser capaz de listar todos os passos existentes num documento formal de caso de uso, contabilizando a ocorrência de passos iguais ao longo do documento.
2. A aplicação deve oferecer a possibilidade de gerar *scripts* tanto para o RFT quanto para o *Selenium* e para isto deve importar os arquivos gravados para os passos.
3. Após a importação dos *scripts*, os casos de teste que ficaram prontos para execução devem ter seus status alterados, para o usuário identificar que estão completamente automatizados.
4. Para os casos de testes que tiveram sua automação finalizada, deve ser habilitada a funcionalidade de executar os casos de teste (*Selenium*) e exportar *script* no caso do RFT.
5. O usuário poderá visualizar os *scripts* gerados, para que os mesmos possam ser copiados para o RFT.
6. Os *scripts*, quando utilizado o *Selenium*, poderão ser executados a partir da TaRGeT e o *log* de execução visualizado.

O processo de desenvolvimento foi precedido de uma análise das ferramentas de automação possíveis de serem usadas pela fábrica de teste interessada no produto. A próxima seção descreve os resultados observados.

5.4 Análise RFT e Selenium

Ambas as ferramentas são largamente usadas no mercado e foram utilizadas por diferentes razões. O *Selenium*, ferramenta *open source* (código aberto) e isenta de compra de licença de uso, pode ser facilmente integrada à TaRGeT, permitindo que a execução dos testes possa ser iniciada a partir dela. Para o caso da fábrica de teste em questão, o uso do RFT, mesmo sendo uma aplicação proprietária, não traz nenhuma restrição, pois há licenças para seu uso. A desvantagem é que as duas ferramentas não poderão ser integradas, ou seja, os *scripts* gerados devem sempre ser copiados para um arquivo do RFT para realizar a execução.

Como qualquer outra ferramenta de automação, um problema em comum apresentado foi a questão do reconhecimento de objetos dinâmicos abordada no capítulo 3. A execução de diversos *scripts* nas duas ferramentas mostrou que o Rational Functional Tester é mais robusto por definir estratégias de detecção de objetos mais avançadas que o *Selenium*, o que não quer dizer que alguns testes não tenham falhado por este motivo.

Segundo (14), o *Selenium* apresenta os seguintes atributos para busca de objetos:

- O ID do elemento
- O nome do elemento
- Uma declaração Xpath
- Um documento de modelo de objetos (DOM)

Identificadores e nomes dos elementos são as opções mais intuitivas de serem utilizadas e o *script* gravado ganha em termos de legibilidade. Essas duas formas são problemáticas quando os ID’s ou nomes são gerados dinamicamente e mudam a cada execução do sistema em teste. As opções Xpath e DOM (*Document Object Model*) realizam o reconhecimento a partir da estrutura da página no momento da gravação do testes, ou seja, para cada interação com um objeto, um caminho que o identifica na

estrutura da página é gerado. (14) A utilização dessas duas últimas abordagens não é muito trivial, visto que é necessário conhecimento das estruturas para montá-las de forma adequada. O RFT utiliza um local centralizado para inserção de objetos de teste chamado de "Mapa de Objetos". Neste repositório os objetos visitados durante a gravação do teste são colocados e, durante a execução do *script*, esta base é consultada para verificação dos atributos de reconhecimento. A figura abaixo mostra os atributos padrões do RFT para um dado objeto:

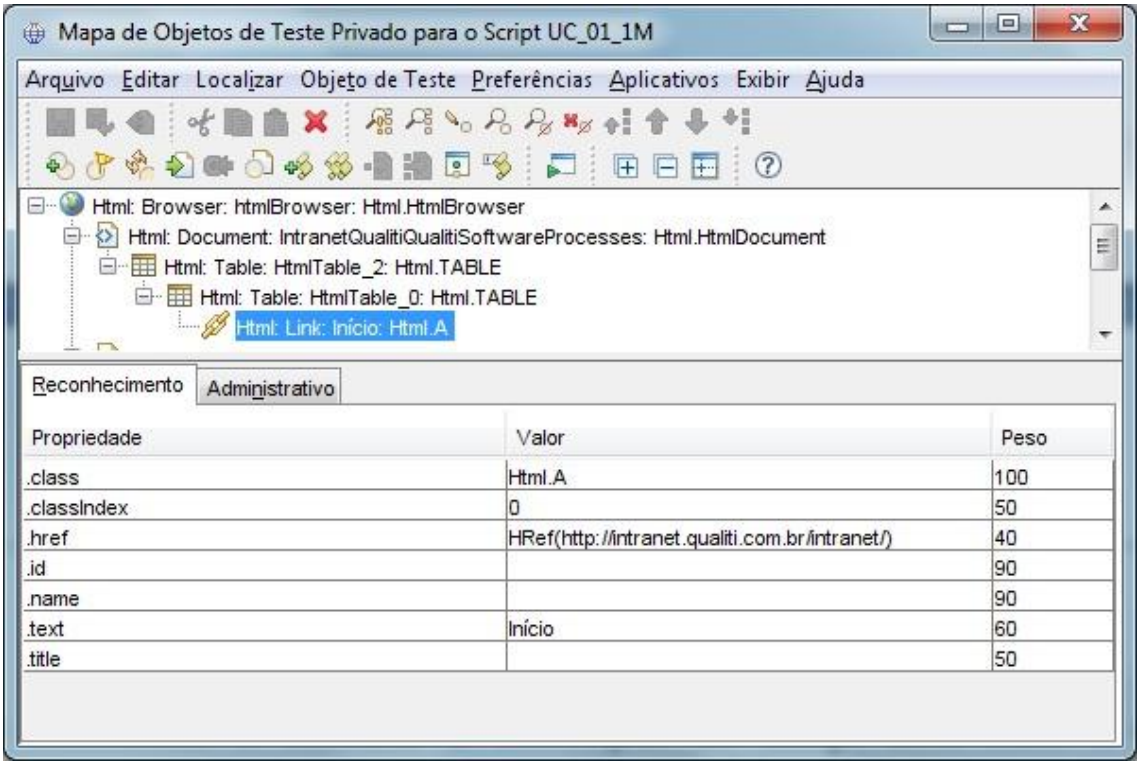


Figura 16 - Reconhecimento de Objetos – Atributos RFT

Baseado nas vantagens e limitações de cada uma dessas ferramentas, decidiu-se construir o *plug-in* de forma que ambas fossem utilizadas. É importante ressaltar que nenhuma delas está adequada à gravação isolada de passos dos casos de teste, por isso o produto deve realizar processamentos sobre os *scripts* ou usar funcionalidades fornecidas pelas ferramentas, para conseguir aplicar o conceito de reuso aos *scripts*

gerados. O ideal seria que a TaRGeT tivesse um módulo de gravação próprio que estivesse sincronizado com sua estrutura de passos, a fim de que a montagem dos testes automáticos fosse um fluxo menos custoso.

A próxima seção mostra como a TaRGeT deve ser utilizada para automatizar os casos de testes gerados.

5.5 Gerando Scripts na TaRGeT

A realização do macro fluxo descrito na figura 15, após o desenvolvimento e integração do *plug-in* à TaRGeT, ocorre da seguinte maneira:

Assumindo que um projeto TaRGeT já tenha sido criado e que haja um documento de caso de uso importado, deve-se proceder à criação dos casos de testes por meio do *On The Fly Generation.* A figura a seguir mostra a interface da TaRGeT para esta etapa.

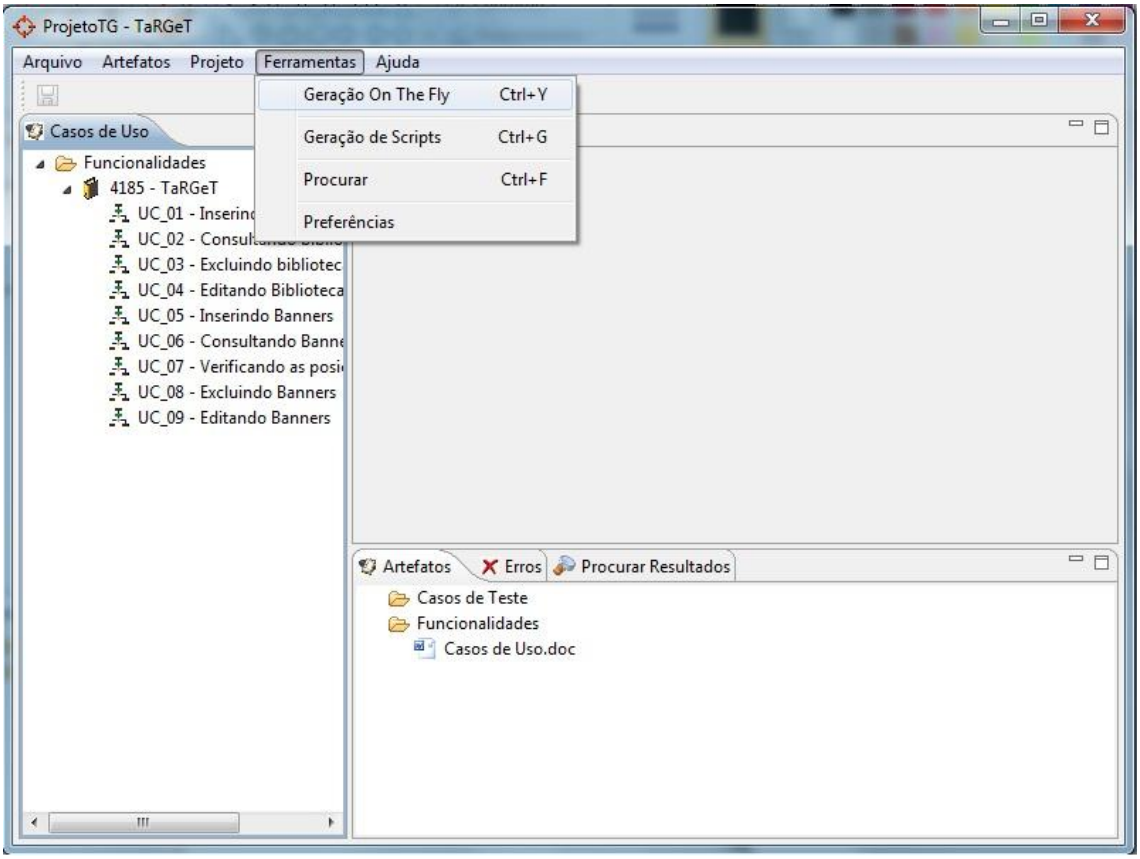


Figura 17 - Gerando Casos de teste – TaRGeT

Após clicar no item selecionado, o editor de geração *On The Fly* é aberto e podem-se aplicar filtros, a fim de selecionar os casos de testes de interesse. Clicando na aba “Casos de teste”, tem-se a visualização da suíte resultante, como mostrado na figura abaixo:

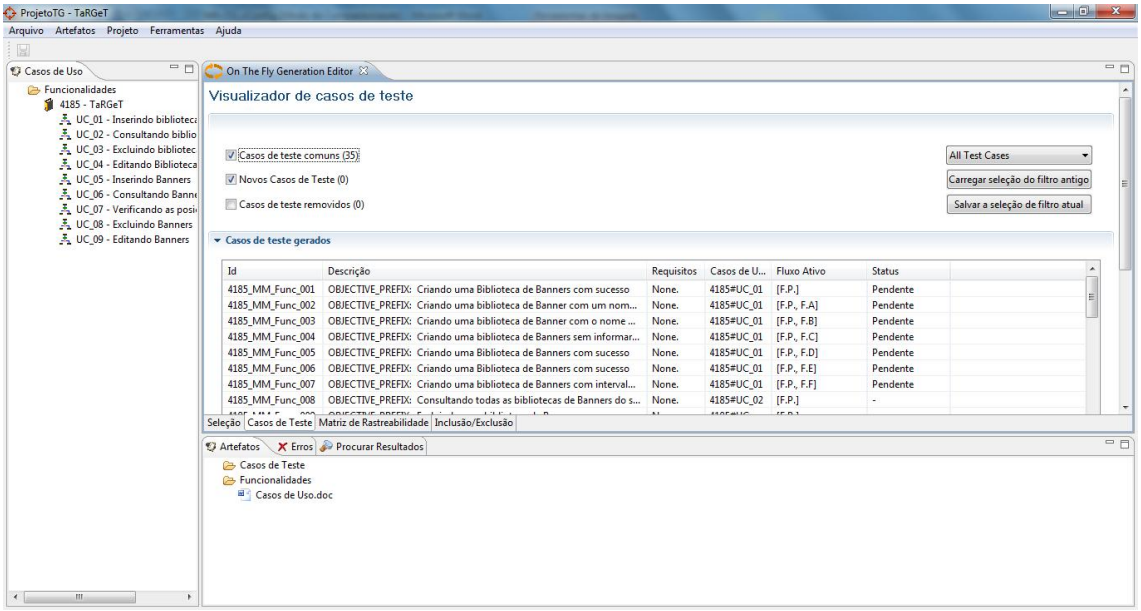


Figura 18 - Visualização Casos de Testes Gerados na TaRGeT

Definida a suíte que se deseja automatizar, pode-se iniciar a funcionalidade

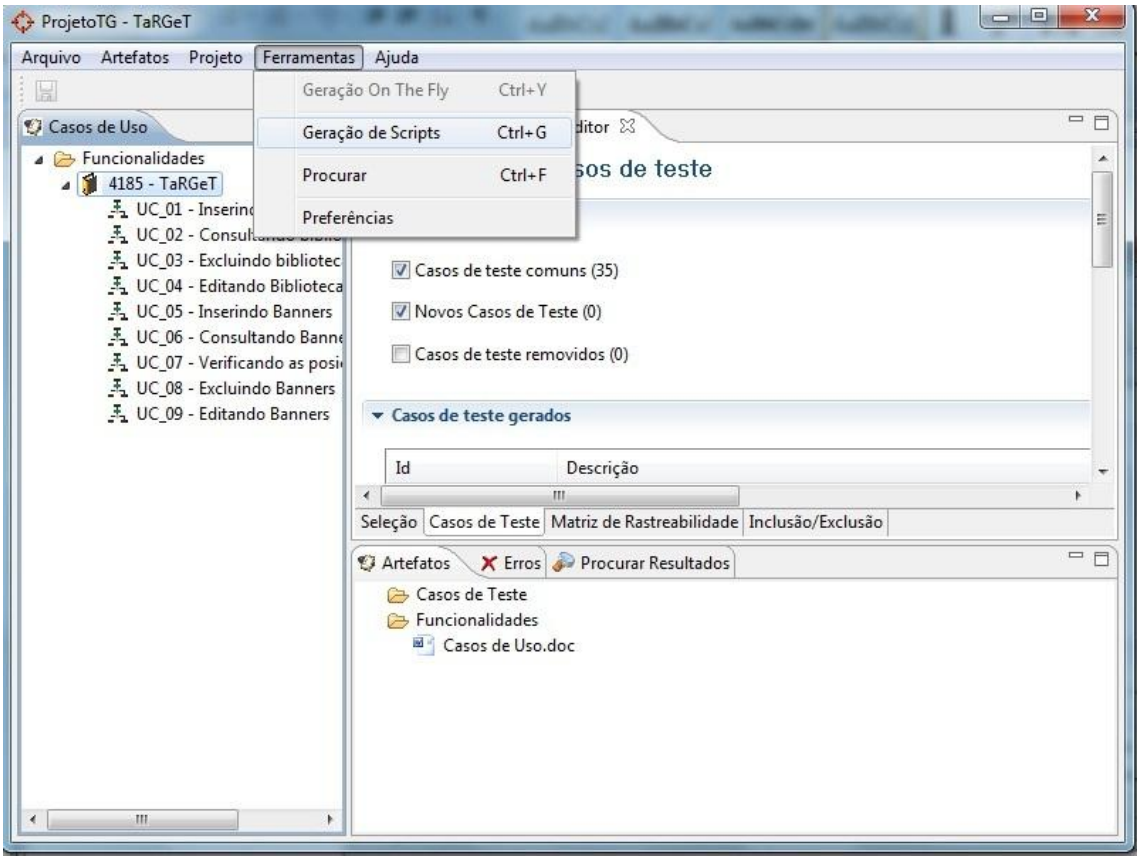


Figura 19 - Iniciando o Gerador de Scripts

O editor de geração de *scripts* inicia e na aba “Segmentos de teste” são listados os passos presentes na suíte gerada anteriormente e o número de vezes que aquele determinado passo é reutilizado em outros casos de teste. Esta visualização é importante para priorizar os passos a serem gravados na ferramenta, objetivando concluir a automação de mais cenários de teste o mais cedo possível. A figura abaixo apresenta a interface da TaRGeT para esta funcionalidade.

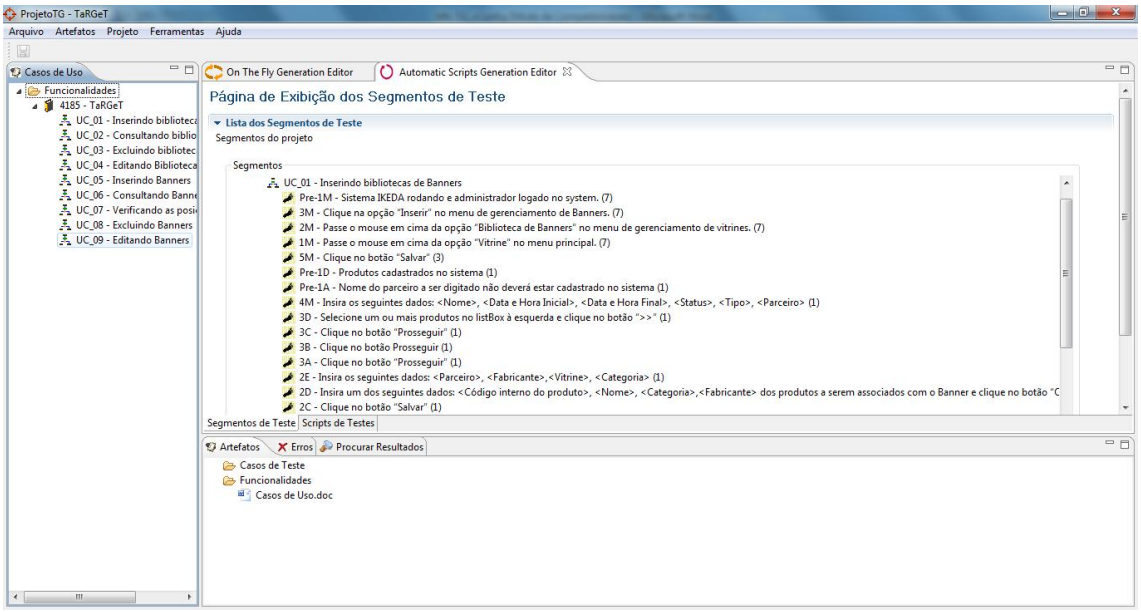


Figura 20 - Segmentos de Teste e número de ocorrências

Tendo uma visualização dos passos a serem gravados, o usuário poderá iniciar o RFT ou o *Selenium* para dar início ao processo de gravação.

Finalizado o processo, os *scripts* dos passos podem ser importados, selecionando-se a aba “Scripts de Testes”, detalhada na figura 21.

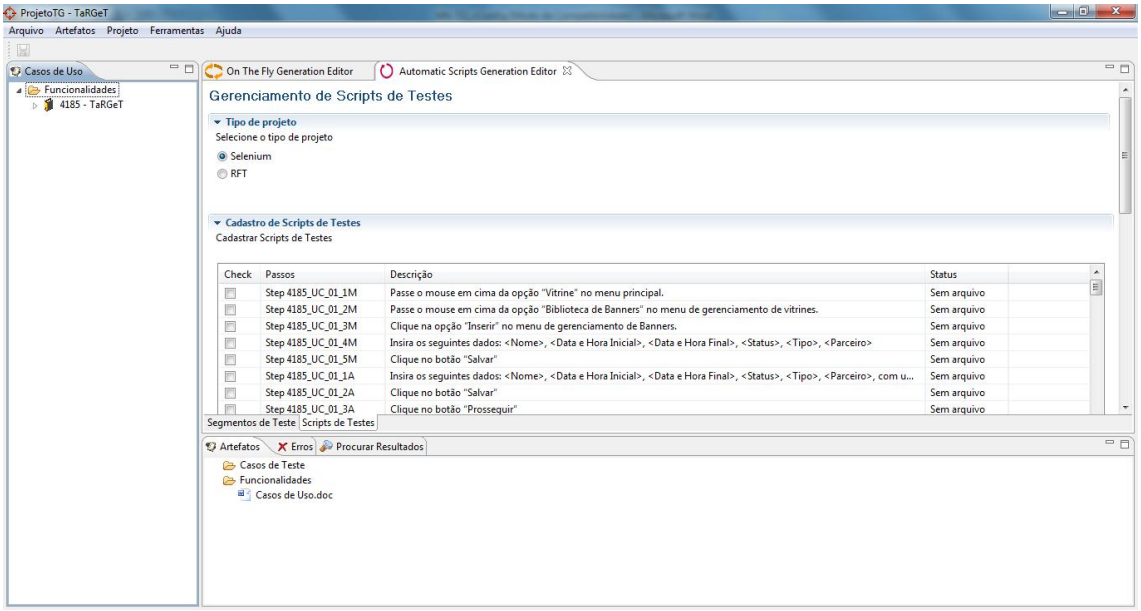


Figura 21 - Interface para Gerenciar os Scripts de Teste

Dependendo da ferramenta escolhida para realizar a gravação, deve-se selecionar o tipo de projeto (RFT ou *Selenium*) antes de proceder com a importação dos *scripts* dos passos. A figura 22 mostra a seleção do projeto para o tipo RFT e a seleção dos passos que terão seus *scripts* importados.

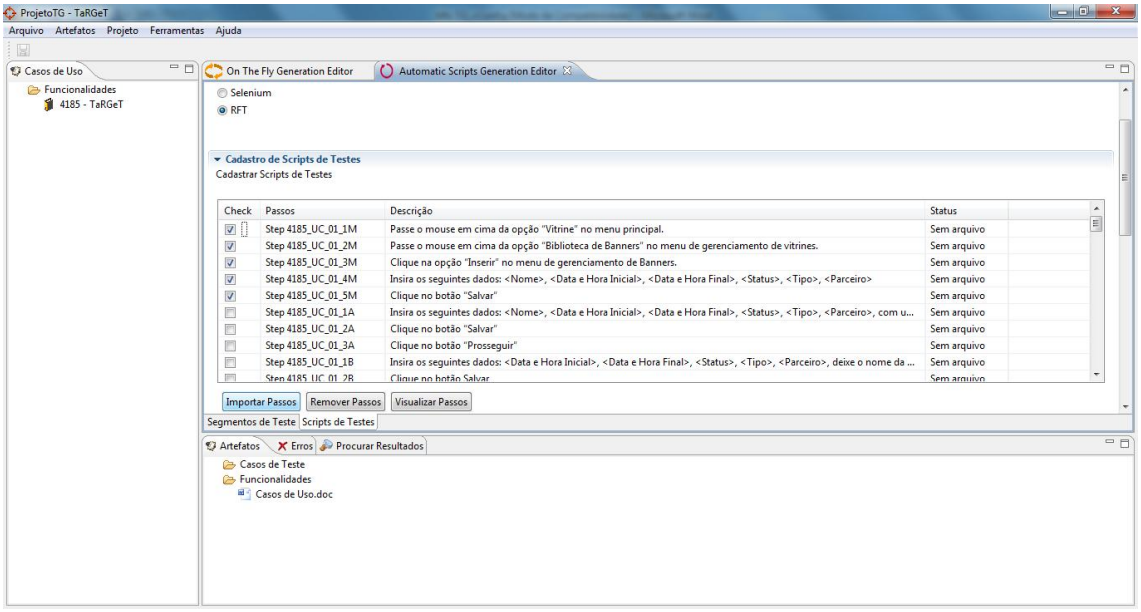


Figura 22 – Seleção do tipo de projeto e passos para importar os scripts

Clicando no botão “Importar Scripts”, mostrado na figura anterior, uma janela de seleção de arquivos é aberta. Após selecionar os arquivos gerados pelas ferramentas (.java) e clicar em “Abrir”, os *scripts* são armazenados em pastas internas do projeto criado e o *status* do passo muda de “Sem Arquivo” para “Importado”.

Nesta fase o reuso de *scripts* acontece, pois todos os testes, que possuem no seu fluxo a ocorrência daquele passo que esta recebendo um *script*, têm seu processo de automação iniciado. A figura 23 ilustra o que foi dito.

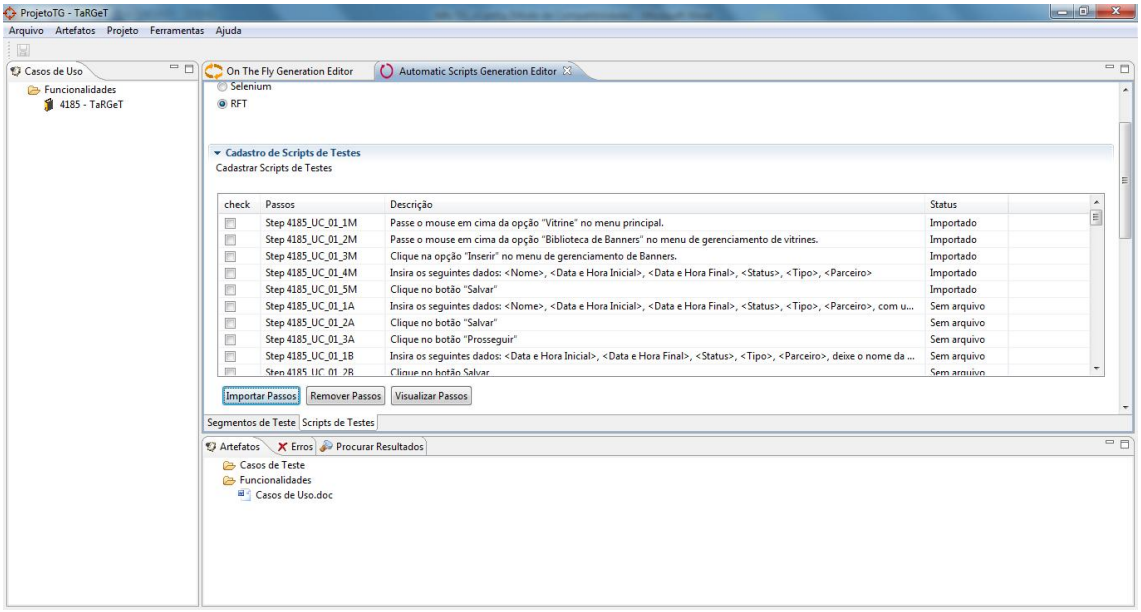


Figura 23 - Importando Scripts TaRGeT

Descendo a barra de rolagem, presente no lado esquerdo da janela da TaRGeT, Observa-se a presença de uma tabela que contém uma visão dos casos de testes que deverão ser automatizados. Note que os *scripts* importados foram suficientes para automatizar o caso de teste mostrado na primeira linha da tabela. Seu *status* mudou de “Não Importado” para “Não Exportado”, significando dizer que o teste já pode ter seu *script* construído. Outros testes, que compartilham os passos que foram automatizados, ficam “aguardando” a importação dos outros passos, para terem seus *status* alterados. A figura a seguir detalha a situação acima.

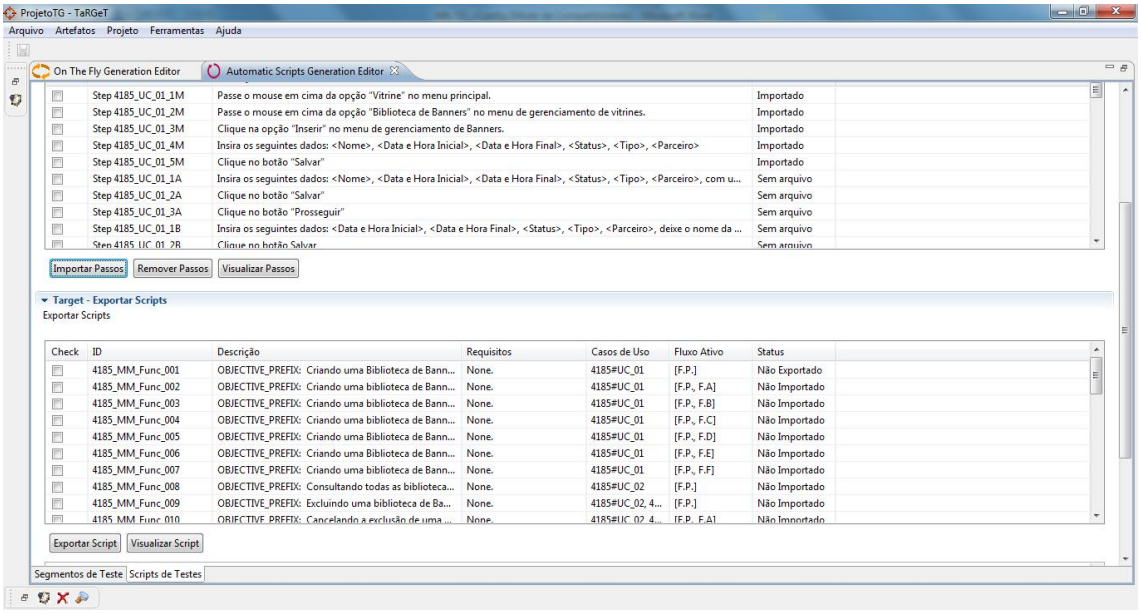


Figura 24 - Status do caso de Teste após importação dos scripts dos passos

Para os casos de teste com o *status* "Não Exportado", é possível gerar os *scripts* por meio da execução da função "Exportar Script", que gera um arquivo contendo o *script* para cada teste selecionado. Na TaRGeT, testes com *scripts* já gerados têm os *status* alterados para "Exportado". Observe a figura abaixo.

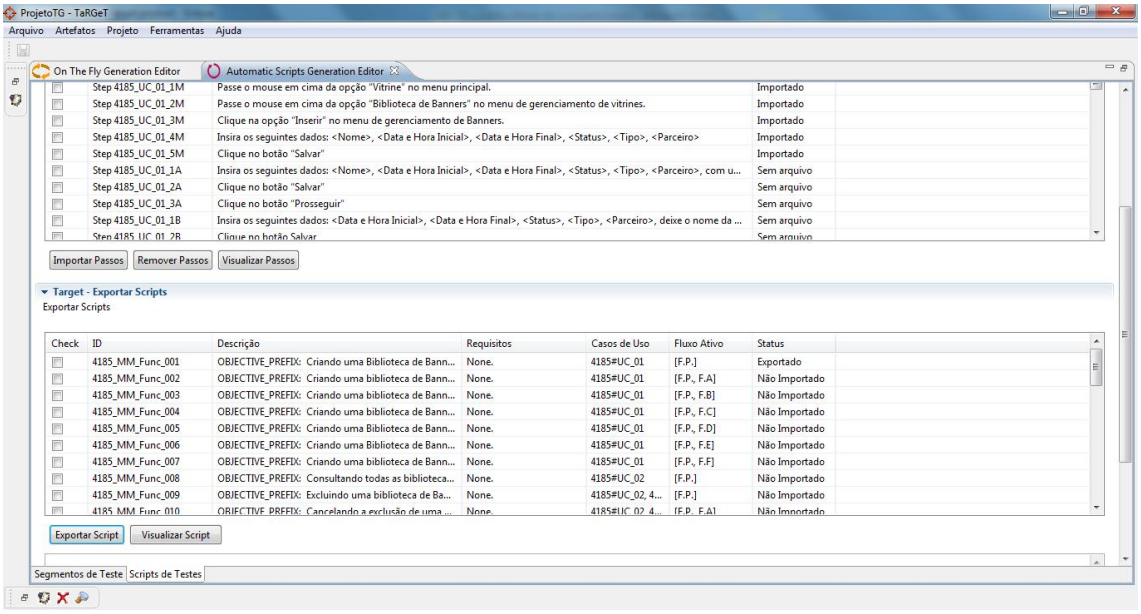


Figura 25 - Status do caso de teste após geração do script

Para visualizar o *script* resultante, basta selecionar o caso de teste desejado e acionar a função “Visualizar Script”. O código é mostrado na TaRGeT e o usuário poderá copiá-lo para o RFT e prosseguir com a execução do teste. A figura 26 apresenta a funcionalidade descrita.

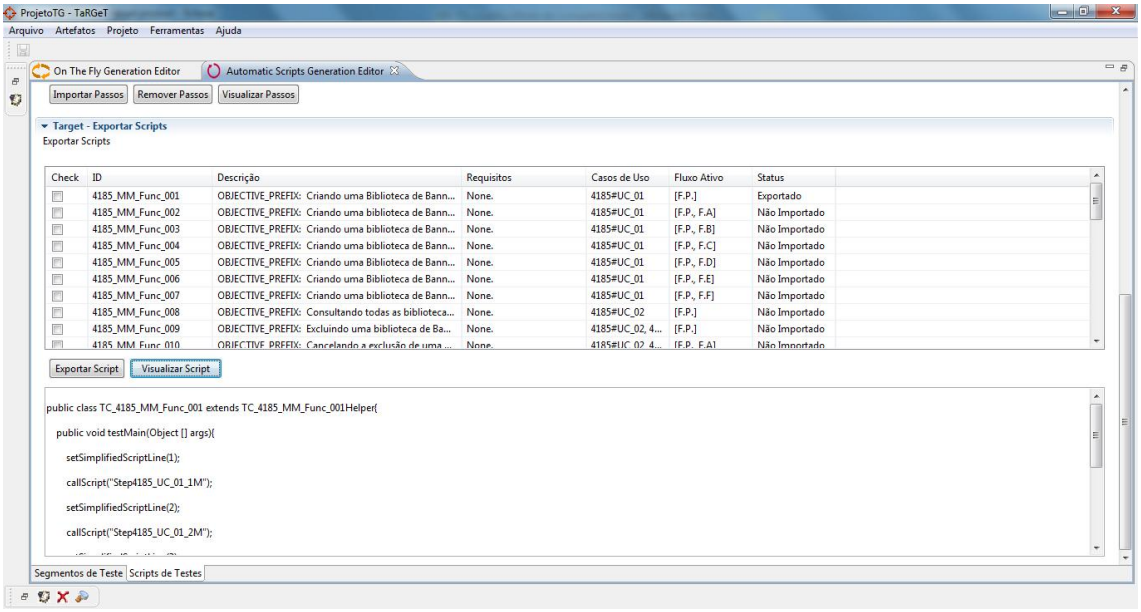


Figura 26 - Visualização do script gerado

O fluxo descrito nesta seção cobriu o funcionamento quando a ferramenta utilizada para a gravação dos passos for o RFT. No caso do *Selenium*, o fluxo é similar até o momento de importação dos *scripts* dos passos. Neste tipo de projeto, os testes que têm seus passos completamente mapeados estão habilitados para execução dentro da própria TaRGeT e a mudança de *status* está relacionada ao resultado da execução. As funcionalidades de exibir “Resultado do Teste” e “Evidências do Teste” referem-se ao detalhamento do fluxo de execução com a exibição do *log* e apresentação de imagens coletadas ao longo da execução do *script*, respectivamente. A figura abaixo mostra as informações após a execução com falha de um caso de teste.

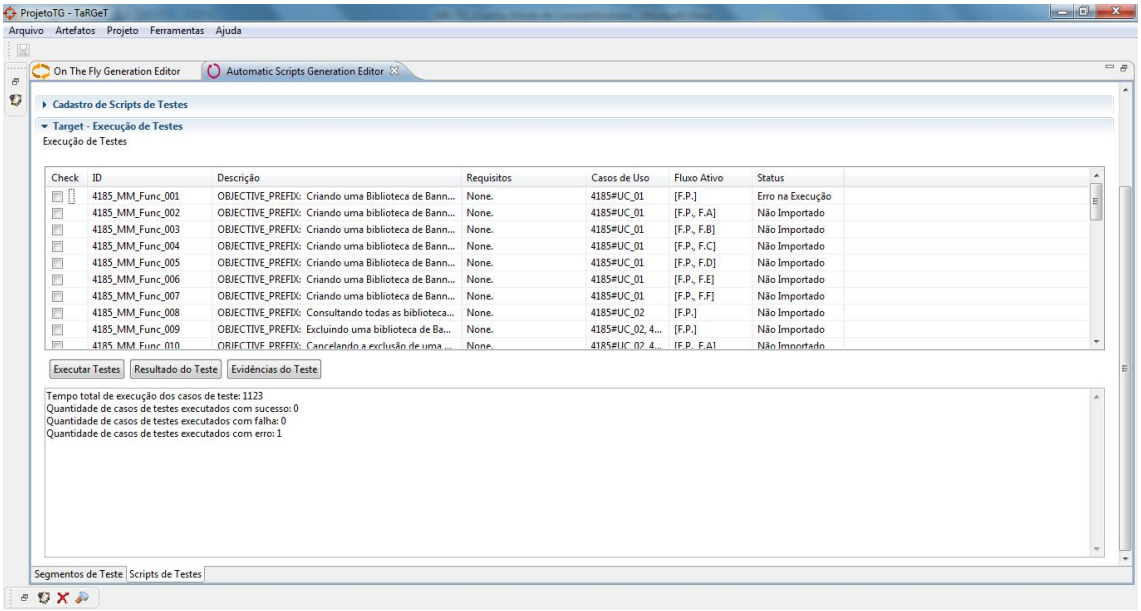


Figura 27 - Funcionalidades disponíveis para scripts do Selenium

Esta seção cobriu o uso das funcionalidades implementadas no *plug-in* de geração de *scripts*, mostrando os elementos desenvolvidos e a fluxo de trabalho a ser seguido pelos usuários.

5.6 Considerações Finais

Criada sobre a plataforma de programação orientada a *plug-ins*, a TaRGeT possibilita que novas funcionalidades sejam adicionadas a sua linha de produtos, através da implementação de pontos de extensão da ferramenta.

A metodologia desenvolvida objetivou inserir uma funcionalidade que impactasse na etapa de automação dos casos de teste e, para isto, duas ferramentas utilizadas numa fábrica de teste foram utilizadas.

O próximo capítulo se dedica a descrever a forma de avaliação do produto criado.

6 AVALIAÇÃO DO PLUG-IN

Este capítulo tem como objetivo descrever a avaliação do produto criado. Na seção 6.1 uma visão geral do capítulo é dada. Na seção 6.2, é realizada uma análise funcional. Na seção 6.3, é mostrada a importância de uma análise quantitativa. Por fim, na seção 6.4 são apresentadas as considerações finais.

6.1 Visão Geral

A proposta de criação e conseqüente implementação de um produto deve estar alinhada com avaliações de usabilidade e eficácia, para que possa ser aceito em uso comercial.

Este capítulo objetiva levantar pontos de análise da ferramenta como forma de validar sua aplicabilidade.

6.2 Análise Funcional

Como o produto foi elaborado tendo em vista as necessidades de uma empresa, sua estrutura e funcionalidades estão de acordo com o processo nela adotado. Apesar de o *plug-in* conseguir realizar a idéia de reuso da TaRGeT, aqui aplicados ao *scripts* de teste, um ponto de fragilidade é o uso de ferramentas externas para gravação dos passos.

O RFT e o *Selenium* atendem ao que foi proposto, mas por conta do ambiente de trabalho não estar totalmente integrado, o uso dessa abordagem pode se tornar cansativo pela mudança constante de contexto entre a ferramenta de automação e a TaRGeT.

Outra questão importante diz respeito à validação, em termos semânticos, dos *scripts* gerados, ou seja, mesmo sendo possível agrupar e reusar os *scripts* dos passos, fica como responsabilidade do automatizador validar se a execução está de acordo com o esperado.

Em termos de usabilidade, é interessante que o produto seja utilizado pela equipe de testes da empresa tratada, a fim de que os pontos de ineficiência sejam levantados. Não foi possível submeter o produto para o ambiente de produção e coletar os resultados a tempo de incluir os relatórios neste trabalho, podendo ser uma atividade relacionada a trabalhos futuros.

A avaliação da versão atual do *plug-in* foi realizada aplicando-se testes com *web sites* de uso geral, como gerenciadores de *e-mail*, fóruns entre outros. Procedeu-se ao fluxo de trabalho descrito no capítulo 5, onde os *scripts* foram gerados e postos à execução. Na maioria dos testes, falhas ocorreram devido ao problema de reconhecimento de objetos, o que indica que é um campo que deve ser explorado para que soluções sejam desenvolvidas e o impacto de sua ocorrência seja minimizado. Esses testes não invalidam a abordagem de junção dos *scripts*, pois em passos cuja interação foi com elementos estáticos, os testes passaram.

6.3 Análise Quantitativa

Como mostrado desde o início, o objetivo geral do *plug-in* é implantar na TaRGeT uma ferramenta que impacte no tempo de automação dos casos de teste por meio do reuso de *scripts*. Analisar quantitativamente e estatisticamente a eficácia da ferramenta, para identificar o grau de redução de tempo da etapa de automação, certamente será uma forma empírica de consolidar a aplicabilidade do processo trazido pelo *plug-in*. Não foi realizado nenhum estudo quantitativo para anexar ao presente trabalho, esta atividade será citada também na seção de trabalhos futuros.

6.4 Considerações Finais

O produto desenvolvido mostrou que a idéia de reuso de *scripts* pode ser aplicada à TaRGeT e, mesmo sua aplicabilidade não tendo sido comprovada em uso operacional, evidencia que melhorias na técnica e no *plug-in* podem ser pesquisadas e implementadas com o intuito do processo de teste se tornar mais eficaz.

O próximo capítulo traz as conclusões e passos futuros para o presente trabalho.

7 CONCLUSÕES E PASSOS FUTUROS

Este capítulo mostra as conclusões sobre este trabalho. A seção 7.1. descreve as principais contribuições deste trabalho. A seção 7.2. aponta as maiores dificuldades encontradas. Finalmente, a seção 7.3. aponta as perspectivas de trabalhos futuros.

7.1 Principais Contribuições

O mapeamento de um documento de caso de uso para passos de casos de testes é uma abordagem bastante útil para garantir que os testes estão de acordo com o proposto no documento de requisitos. Utilizar a TaRGeT para gerar esses testes traz um ganho interessante no tempo de arquitetura dos cenários, tornando a ferramenta uma forma de melhorar uma parte do processo de teste. O *plug-in* proposto visa adicionar à TaRGeT um componente que a torne mais importante para o processo, melhorando a fase de automação e execução dos cenários criados. A consolidação do uso do *plug-in* objetiva fazer com que fábricas de testes, que porventura a utilizem, apresentem ganhos de performance no fluxo de negócio e conseqüentemente ganho de mercado.

7.2 Dificuldades Encontradas

Para tornar o *plug-in* aplicável no ambiente para o qual foi criado, foi necessário adaptar o funcionamento das ferramentas de gravação de *scripts* utilizadas para prover o meio de execução dos testes. Tanto o *Selenium*, quanto o RFT não realizam o reuso de *scripts* de uma maneira “natural” e um dos desafios foi manipular o código gerado pelas ferramentas, para que fossem montados de uma forma que a execução não apresentasse erros. Outra dificuldade, talvez a maior, foi a realização dos testes com sistemas reais, pelo motivo da ocorrência de objetos dinâmicos. Inicialmente, a execução falha de

alguns testes deixou dúvida se o problema vinha da montagem dos *scripts* dos passos ou do não reconhecimento de um objeto de teste e só foi solucionada quando fluxos que interagiam com objetos estáticos foram criados, mostrando que o algoritmo para junção das partes dos casos de teste não estava errado.

7.3 Trabalhos Futuros

Uma continuidade interessante deste trabalho seria a coleta de informações num ambiente de testes que utilize automação, inserindo a TaRGeT juntamente com o *plug-in* desenvolvido, para que haja um embasamento real de sua usabilidade e eficácia, ou seja, um estudo quantitativo de sua aplicação num ambiente real de automação, para verificar o ganho no processo com o uso da ferramenta.

Fazer com que a TaRGeT possua seu próprio instrumento de gravação de *scripts*, já adaptado para o reuso de passos gravados evitaria que ferramentas de terceiros fossem usadas e o esforço de adaptá-las para atender ao reuso seria anulado.

Desenvolver modelos para reconhecimento de objetos dinâmicos traria uma contribuição significativa, caso a TaRGeT passasse a ter sua própria ferramenta de gravação.

Referências

(1) SOMMERVILLE, I. Software Engineering. Edinburgh: Addison-Wesley, 1995.

(2) BURNSTEIN, Ilene. Practical Software Testing. New York: Springer, 2002.

(3) FEWSTER, Mark; GRAHAM, Dorothy. Software Test Automation: effective use of test execution tools. Edinburgh: ACM Press, 1999.

(4) FERREIRA. et al. TaRGeT: a Model Based Product Line Testing Tool. In: CONGRESSO BRASILEIRO DE SOFTWARE, 2010. Anais... Salvador: CBSoft, 2010.

(5) SYLLABUS. Florida: American Software Testing Qualifications Board , 2007. 77p.

(6) APOSTILA do curso de Testes de Software da Qualiti Software Processes.

(7) KANER, C; FALK, J; NGUYEN, H: Testing Computer Software. 2 ed. [S.l]: Willey, 1999.

(8) FANTINATO, M. et al. AutoTest: Um *framework* reutilizável para a automação de teste funcional de software. Disponível em: http://www.sbc.org.br/bibliotecadigital/download.php?paper=255. Acesso em: 06 de nov. 2010.

(9) LAUKKANEN, P. Data-driven and keyword-driven test automation Frameworks. Disponível em: <http://eliga.fi/Thesis-Pekka-Laukkanen.pdf>. Acesso em: 06 de nov. 2010.

(10) IBM Rational Functional Tester documentation. Disponível em: < http://public.dhe.ibm.com/software/rational/web/datasheets/rft.pdf>. Acesso em: 06 de nov. 2010.

(11) SELENIUM Documentation. Disponível em: < http://seleniumhq.org/docs/01_introducing_selenium.html>. Acesso em: 06 de nov. 2010.

(12) ANISZCZYC, C. Plug-in Development. Disponível em: < http://www.ibm.com/developerworks/library/os-eclipse-plugindev1/>; Acesso em: 08 de nov. 2010.

(13) GAMMA, E; BECK, K: Contributing to eclipse: principles, patterns and plug-ins. [S.l.: s.n.], 2004. (The Eclipse Series).

(14) SELENIUM Test Design Considerations. Disponível em: <http://seleniumhq.org/docs/06_test_design_considerations.html>. Acesso em: 06 de nov. 2010.