GERAÇÃO AUTOMÁTICA DE SCRIPTS DE TESTE UTILIZANDO O SELENIUM

Trabalho de Conclusão de Curso

Engenharia da Computação

Tiago Xavier Lopes da Silva

Orientador: Prof. M.Sc. Gustavo H. P. Carvalho

Tiago Xavier Lopes da Silva

GERAÇÃO AUTOMÁTICA DE SCRIPTS DE TESTE UTILIZANDO O SELENIUM

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco - Universidade de Pernambuco

Orientador:
Prof. M.Sc. Gustavo H. P. Carvalho

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

Recife - PE, Brasil

05 de dezembro de 2011

Declaro que revisei o Trabalho de Conclusão de Curso sob o título *"GERAÇÃO AUTOMÁTICA DE SCRIPTS DE TESTE UTILIZANDO O SELENIUM"*, de autoria de *Tiago Xavier Lopes da Silva*, e que estou de acordo com a entrega do mesmo.

Recife, _____ / _____ / _____

Prof. M.Sc. Gustavo H. P. Carvalho
Orientador

Resumo

Teste de software é um importante elemento para verificar corretudi de um sistema. No entanto, esta é uma atividade complexa e custosa do processo de desenvolvimento de software. Com o intuito de minimizar os custos e potencializar esta atividade, ela costuma ser parcialmente automatizada. Porém a automação dos testes gera a outro ponto: como gerar automaticamente os scripts de teste. Visando oferecer uma possível solução a essa questão, este trabalho desenvolveu uma ferramenta de automação de testes baseada no Selenium. Nesta ferramenta, os scripts de teste em Selenium são gerados automaticamente a partir de texto em linguagem natural, em função de mapeamentos definidos previamente pelo usuário.
Palavras-chave: teste funcional, automação de teste, Selenium, linguagem natural.

Abstract

Software testing is an important element to check correctness of a system. However, this is a complex and costly process of software development. In order to minimize costs and maximize this activity, it tends to be partially automated. But the automation of testing generates another point: how to automatically generate test scripts. Aiming to offer a possible solution to this issue, this paper developed an automated testing tool based on Selenium. In this tool, Selenium test scripts are generated automatically from natural language text, according to mappings defined in advance by the user.

Keywords: Functional Testing, Automation Testing, Selenium.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
OCL	Object Constraint Language
OMG	*Object Management Group*
RUP	*Rational Unified Process*
SPACES	*Specification based Component tester*
SWEBOK	*Software Engineering Body of Knowledge*
TDE	*Test Development Environment*
UCT	*Use Case Tester*
UML	*Unified Modeling Language*
XMI	*XML Metadata Interchange*
XML	*Extensible Markup Language*

1 Introdução

1.1 Qualificação do Problema

Segundo o SWEBOK (2004), teste de software consiste na verificação dinâmica do comportamento de um programa através de um conjunto finito de casos de teste. Este conjunto deve ser adequadamente selecionado a partir do infinito número de possibilidades do comportamento esperado especificado.

Atividades de teste devem ser cuidadosamente planejadas para fornecer um nível adequado de confiabilidade. Com o intuito de realizar corretamente os requisitos de teste dentro das condições especificadas para assim contribuir na qualidade de software. Com o intuito de atingir esse objetivo, estas atividades concentram um conjunto de técnicas, critérios e métodos que fornecem ao projetista uma abordagem sistemática para realizar o teste do sistema em questão.

A atividade de teste pode ser dividida em quatro níveis \estágios. Cada um deles está mais direcionado a uma fase do desenvolvimento de software, sendo eles:

Testes Unitários : serve para garantir que uma funcionalidade especificada nos requisitos, seja implementada adequadamente.

Testes de Integração : aplicado para verificar e garantir que o software desenvolvido consiga se comunicar adequadamente com outros softwares. Neste nível existe duas técnicas ou métodos:

- **Abordagem Top-Down** : utiliza uma abordagem incremental e serve para identificar problemas de forma antecipada.
- **Abordagem Botton-Up** : ao contrário do Top-Down, detecta algum problema no escopo da arquitetura tardiamente.

Testes de Sistema : verifica toda a funcionalidade do sistema e a precisão dos mesmos.

Teste de Aceitação : visa garantir que o sistema está apto para ser implantado em produção.

Segundo o *Rational United Process* – RUP (RUP, 2011) – estas técnicas são baseadas em:

- Funcionalidades: técnica de teste em que o componente de software a ser testado é abordado como se fosse uma caixa-preta, ou seja, não se considera o comportamento interno do mesmo (Figura 1). Nesta técnica, dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado esperado, previamente conhecido. Haverá sucesso no teste se o resultado obtido for igual ao resultado esperado. O componente de software a ser testado pode ser um método, uma função interna, um programa, um componente, um conjunto de programas e/ou componentes ou mesmo uma funcionalidade. A técnica de teste funcional é aplicável a todos os níveis de teste (PRESSMAN, 2005);

Figura 1: Técnica de Teste Funcional.

- Estruturas: apoia-se essencialmente na implementação, no fluxo de controle e em informações do fluxo de dados. Esta abordagem busca garantir que o produto esteja estruturalmente sólido e funcione corretamente (Figura 2).

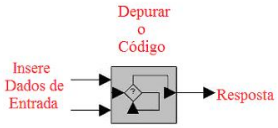

Figura 2: Técnica de Teste Estrutural.

- Erros: técnica que utiliza informações sobre os tipos de erros mais frequentes no processo de desenvolvimento de software para derivar os casos de teste. A ênfase da técnica está nos erros que o programador ou projetista pode cometer durante o desenvolvimento e nas abordagens que podem ser usadas para detectar a sua ocorrência.

Além disso, pode-se usar uma combinação das técnicas supra citadas.

O teste de software é considerado como uma atividade que consome entre 30 e 50% do esforço total do desenvolvimento (SOMMERVILLE, 2007), e um erro descoberto na fase de implantação provoca um acréscimo aproximado de 60% nos custos do projeto (TORRES, 2011). Visando melhorar a amplitude do teste, sua capacidade de repetição e reduzir custos, costumam ser aplicadas técnicas de automação de testes. A automação de testes é definida pelo uso de software para controlar a execução do teste de software, a comparação dos resultados esperados com os resultados reais, a configuração das pré-condições de teste e outras funções de controle e o relatório de teste. Além de consumir tempo e em alguns casos mais ainda do que a própria execução manual. Portanto, antes de decidir automatizar um caso de teste, deve-se avaliar se o custo de automação é inferior ao custo da quantidade esperada de execuções manuais.

Uma forma de reduzir os custos da atividade de testes é a partir da automação dos testes usando alguma ferramenta apropriada. No caso dos testes funcionais, a maioria das ferramentas disponíveis se baseia na gravação de todas as ações executadas por um usuário, gerando um script com os passos realizados. Dessa forma, as ações gravadas podem ser facilmente re-executadas, permitindo assim a automação do teste. Além disso, o script gerado pode ser facilmente alterado, permitindo modificações nos testes gravados sem a necessidade de re-execução do software e nova captura (NETO P. S.; RESENDE, 2007). Como exemplo desse tipo de ferramenta temos o Selenium (SELENIUM, 2011), que pode ser usado para automatizar testes funcionais em aplicações web.

Ainda que automação de testes seja um modelo viável para a otimização da atividade de teste, ela demanda um novo problema: a codificação e manutenção dos scripts de teste que, por sua vez são, em muitas vezes, elaborados manualmente e exigem uma maior especialização do testador.

Assim sendo, uma solução interessante para a situação apresentada seria a existência de instrumentos que permitissem a geração automática de scripts de teste, diminuindo ainda mais os custos desta atividade. Tendo este problema em vista, este trabalho tem como objetivo de pesquisa: **como gerar automaticamente scripts de teste, utilizando Selenium a partir da especificação de casos de uso de um sistema?**

1.2 Objetivos

Este trabalho tem como objetivo elaborar uma ferramenta de desenvolvimento com o intuito de proporcionar um ambiente integrado que permita gerar, automaticamente, scripts de testes Selenium a partir da descrição em linguagem natural de casos de uso de um dado sistema, e que também permita executá-los e analisar os resultados dessa execução.

1.2.1 Objetivos Específicos

- Estabelecer um mapeamento direto entre frases (linguagem natural) e comandos Selenium.
- Implementar um mecanismo de identificação de cenários a partir da descrição dos casos de uso;
- Implementar um mecanismo de geração de casos de teste a partir dos cenários identificados anteriormente;
- Avaliar, preliminarmente, o uso da ferramenta no teste de uma aplicação real.

1.3 Resultados e Impactos Esperados

Os resultados esperados são:

1. Um ambiente integrado de planejamento, elaboração e execução de testes;
2. Um recurso que permita um fácil reuso de scripts de teste Selenium;
3. Uma ferramenta de geração automática de scripts de teste.

Os impactos esperados são:

1. Amplificação do reuso de código, no tocante a scripts de teste;
2. Redução no custo das atividades de teste, uma vez que o código será gerado de forma automática e pode ser reutilizados;

1.4 Estrutura da Monografia

O restante do trabalho está definido da seguinte forma:

Capítulo 2: apresenta a base teórica para o entendimento deste trabalho . Nele é dada uma breve explanação sobre os tipos de teste, suas atividades sobre Selenium.

Capítulo 3: neste capítulo é discutida a metodologia e a estratégia de ação empregada para se chegar ao objetivo deste trabalho.

Capítulo 4: neste capítulo são apresentados os resultados obtidos pelo trabalho desenvolvido neste projeto.

Capítulo 5: apresentada a conclusão e sugere trabalhos futuros relacionados ao contexto deste trabalho.

2 Referencial Teórico

Este capítulo apresenta os principais conceitos relacionados ao desenvolvimento deste estudo, assim como outros trabalhos relacionados a esta iniciativa.

2.1 Conceitos Chaves

Esta seção apresenta os conceitos chaves utilizados neste trabalho.

2.1.1 Testes de Software

O teste de software é uma atividade que consiste na descoberta de defeitos que possam ter sido introduzidos em qualquer fase do desenvolvimento ou manutenção de sistemas de software e que, em geral, são decorrentes de omissões, inconsistências ou mau entendimento das especificações ou dos requisitos dos sistemas (MCGREGOR; SYKES, 2001).

Existem três técnicas que podem ser aplicadas na prática de testes de software:

- Teste Funcional: checa se um programa está de acordo com a sua especificação, independentemente do código fonte. Teste funcional é também conhecido como teste caixa-preta ou teste comportamental.
- Teste Estrutural: baseia-se na estrutura do objeto testado. Requer, portanto, acesso completo ao código fonte do programa para ser realizado. É também conhecido por teste caixa-branca.
- Combina as duas técnicas de teste anteriores. Definido por alguns autores como teste caixa-cinza.

A prática de testes funcionais é frequentemente melhor aceita do que a prática de testes estruturais pois a mesma pode ser aplicada antes mesmo da disponibilização do código, deste modo, contribuindo para o aperfeiçoamento dos artefatos de desenvolvimento.

Aas seções seguintes descrevem mais detalhes sobre as técnicas de teste funcional e estrutural.

2.1.1.1 Técnica Funcional

O teste funcional também é conhecido como teste caixa preta pelo fato de tratar o software como uma caixa cujo conteúdo é desconhecido e da qual é possível apenas visualizar o lado externo. Nesta técnica são verificadas as funções do sistema sem se preocupar com os detalhes de implementação.

O teste funcional envolve dois passos principais: identificar as funções que o software deve realizar e criar casos de teste capazes checar se essas funções estão sendo realizadas pelo software (PRESSMAN, 2005). As funções que o software deve possuir são identificadas a partir de sua especificação. Assim, uma especificação bem elaborada e de acordo com os requisitos do usuário é essencial para esse tipo de teste.

Ainda segundo PRESSMAN (2005), alguns exemplos de critérios utilizados na concepção de testes funcionais são:

Particionamento em Classes de Equivalência: a partir das condições de entrada de dados identificadas na especificação, divide-se o domínio de entrada de um programa em classes de equivalência válidas e inválidas. Em seguida, seleciona-se o menor número possível de casos de teste, baseando-se na hipótese que um elemento de uma dada classe seria representativo da classe toda, sendo que para cada uma das classes inválida deve ser gerado um caso de teste distinto. O uso de particionamento permite examinar os requisitos de forma sistemática e restringir o número de casos de teste existentes. Alguns autores também consideram o domínio de saída do programa para estabelecer as classes de equivalência.

Análise do Valor Limite: é um complemento ao critério de Particionamento em Classes de Equivalência, sendo que os limites associados às condições de entrada são exercitados de forma mais rigorosa. Ao invés de selecionar-se qualquer elemento de uma classe, os casos de teste são escolhidos nas fronteiras das classes, pois nesses pontos se concentra um grande número de erros de desenvolvimento. O espaço de saída do programa também é particionado e são exigidos casos de teste que produzam resultados nos limites dessas classes de saída.

Grafo de Causa-Efeito: os critérios anteriores não exploram combinações das condições de entrada. Este critério estabelece requisitos de teste baseados nas possíveis

combinações das condições de entrada. Primeiramente, são levantadas as possíveis condições de entrada (causas) e as possíveis ações (efeitos) do programa. A seguir é construído um grafo relacionando as causas e efeitos levantados. Esse grafo é convertido em uma tabela de decisão a partir da qual são derivados os casos de teste.

2.1.1.2 Técnica Estrutural

Na técnica de teste estrutural, também conhecida como teste caixa branca (em oposição ao nome caixa preta), os aspectos de implementação são fundamentais na escolha dos casos de teste. O teste estrutural baseia-se no conhecimento da estrutura interna da implementação. Em geral, a maioria dos critérios dessa técnica utiliza uma representação de programa conhecida como grafo de fluxo de controle ou grafo de programa. Um programa P pode ser decomposto em um conjunto de blocos disjuntos de comandos; a execução do primeiro comando de um bloco acarreta a execução de todos os outros comandos desse bloco, na ordem dada. Todos os comandos de um bloco, com exceção do primeiro, têm um único predecessor e exatamente um único sucessor, exceto possivelmente o último comando.

A representação de um programa P como um grafo de fluxo de controle G = {N;E; s} – onde N é definido pelo conjunto de nós do grafo e E seu conjunto de arestas – consiste em estabelecer uma correspondência entre nós e blocos e em indicar possíveis fluxos de controle entre blocos através dos arcos. Um grafo de fluxo de controle é portanto um grafo orientado, com um único nó de entrada $s \in N$ e um único nó de saída, no qual cada vértice representa um bloco indivisível de comandos e cada aresta representa um possível desvio de um bloco para outro. Cada bloco tem as seguintes características:

1. uma vez que o primeiro comando do bloco é executado, todos os demais são executados sequencialmente;

2. não existe desvio de execução para nenhum comando dentro do bloco. A partir do grafo de programa podem ser escolhidos os componentes que devem ser executados, caracterizando assim o teste estrutural. Segundo (PRESSMAN, 2005) os critérios utilizados na criação de testes estruturais são, em geral, classificados em:

   Critérios Baseados em Fluxo de Controle: utilizam apenas características de controle da execução do programa, como comandos ou desvios, para determinar quais estruturas são necessárias. Os critérios mais conhecidos dessa classe são

Todos-Nós – exige que a execução do programa passe, ao menos uma vez, em cada vértice do grafo de fluxo, ou seja, que cada comando do programa seja executado pelo menos uma vez; **Todos-Arcos** – requer que cada aresta do grafo, ou seja, cada desvio de fluxo de controle do programa, seja exercitada pelo menos uma vez; e **Todos-Caminhos** – requer que todos os caminhos possíveis do programa sejam executados.

Critérios Baseados em Fluxo de Dados: utilizam informações do fluxo de dados do programa para determinar os requisitos de teste. Esses critérios exploram as interações que envolvem definições de variáveis e referências a tais definições para estabelecerem os requisitos de teste.

Critérios Baseados na Complexidade: utilizam informações sobre a complexidade do programa para derivar os requisitos de teste. Um critério bastante conhecido dessa classe é o **Critério de McCabe**, que utiliza a complexidade cíclica do grafo de programa para derivar os requisitos de teste. Essencialmente, esse critério requer que um conjunto de caminhos linearmente independentes do grafo de programa seja executado (PRESSMAN, 2005).

2.1.2 Automação de Testes

Automação de teste é o uso de software para controlar a execução do teste de software, a comparação dos resultados esperados com os resultados reais, a configuração das pré-condições de teste e outras funções de controle e relatório de teste. De forma geral, a automação de teste pode ser começada a partir de um processo manual de teste já estabelecido e formalizado.

Apesar do teste manual de software permitir encontrar vários erros em uma aplicação, é um trabalho maçante e consome bastante tempo. Também, pode não ser efetivo na procura de classes específicas de defeitos. A automação é o processo de escrita de um programa de computador para realizar o teste. Uma vez automatizado, um grande número de casos de teste podem ser validados rapidamente. As vantagens da automação tornam-se mais evidentes para os casos dos produtos que possuem longa vida no mercado, porque mesmo pequenas correções no código da aplicação podem causar a quebra de funcionalidades que antes funcionavam.

A automação envolve testes de caixa-preta, em que o desenvolvedor não possui conhecimento sobre a estrutura interna do sistema, ou caixa-branca, em que há pleno conhecimento da estrutura interna. Para os dois casos, a cobertura de teste é determinada

respectivamente pela experiência do desenvolvedor ou pela métrica de cobertura de código.

A automação de testes tem grande relevância na qualidade do software, pois tem maiores condições de assegurar que um software irá se comportar de forma adequada quando submetido ao uso depois de instalado na produção. Para tanto, os conceitos de qualidade de software devem ser utilizados durante todo o ciclo de desenvolvimento do sistema, assegurando que cada fase do ciclo de desenvolvimento seja homologada com mais segurança, possibilitando a não propagação de um defeito até o final do desenvolvimento.

Outro fator a considerar é que através da automação, o tempo de execução do teste ocorre em proporções bem menores em relação ao tempo executado por um processo de teste manual, onde em algumas situações a equipe não tem tempo suficiente para realizar todos os casos de teste planejados.

Portanto, o custo benefício da automação de teste pode ser verificado sob alguns aspectos:

- Ganho na avaliação da qualidade do software, pois permite realizar maior quantidade de testes em menos tempo;
- Maior cobertura de testes;
- Mais tempo para que a equipe possa realizar outras atividades do processo de teste, como detalhar e elaborar com mais cuidado os casos de testes complexos.

2.1.3 Selenium

O Selenium (2011) é uma ferramenta usada para criação de testes automatizados para aplicações web. O Selenium possui um conjunto de ferramentas de software, cada uma com uma abordagem diferente para apoiar a automação de testes. Todo o conjunto de ferramentas resulta em um rico conjunto de funções de teste especificamente orientadas para as necessidades de testes de aplicações web de todos os tipos. Estas operações são altamente flexíveis, permitindo muitas opções para a localização de elementos de interface do usuário e comparação dos resultados dos testes contra o comportamento esperado do aplicativo real. Uma das principais características do Selenium é o apoio para a execução de testes de em múltiplos navegadores.

O Selenium pode ser dividido em dois modos diferentes:

Selenium Core: Neste modo os teste são executados a partir do Web Server que a

aplicação estiver sendo executada. Os testes são escritos previamente em tabelas HTML e o TestRunner (Gerenciador de execução dos testes) é responsável pela execução dos testes (MANUAL..., 2011).

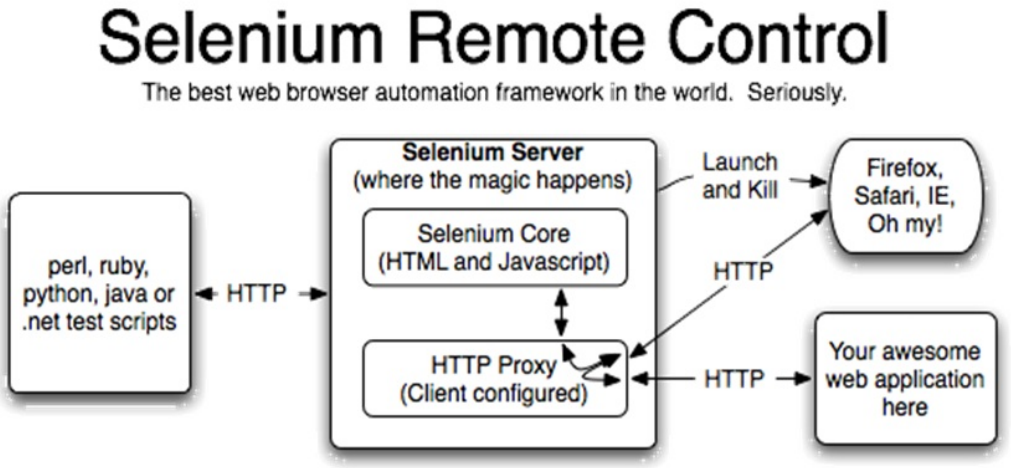

Figura 3: Selenium RC.
[Fonte: Fonte: Cristiano Caetano, Automação e Gerenciamento de Testes]

Selenium RC(Remote Control): Neste modo os testes são dirigidos por uma das linguagens suportadas pelo Selenium (Ruby, Java, C#, Pyton, Php e Perl)

Para testar um sistema, precisamos ser capazes de interagir com ela e fazer afirmações sobre os resultados. Quando o sistema que queremos testar possui uma interface web, é preciso interagir com a página / browser e fazer afirmações sobre o resultado dessas interações.

As interações mais comuns são realizadas através dos seguintes métodos na classe DefaultSelenium:

- open(String url) – método que carrega no navegador a url informada como parâmetro.
- click(String locator) – método que aciona evento de click sobre o elemento informado como parâmetro.
- type(String locator, String value) – método que escreve o valor do parâmetro "value" no elemento "locator", informados como parâmetro.

- select(String locator, String optionLocator) – método que seleciona a opção "optionLocator" em um *drop-down* "locator", informados como parâmetro.
- check(String locator) – método que
- waitForPageToLoad(String timeoutInMilliseconds) – método que aguarda pelo carregamento de uma url por um dado período de tempo.

Agora que temos uma forma de interagir, é preciso ser capaz de obter informações sobre uma página. DefaultSelenium tem muitos métodos para obter informações sobre uma página. Os mais utilizados são:

- getTitle () – método que captura o titulo da janela aberta no navegador.
- getText (String locator) – método que captura o texto de um elemento informado como parâmetro.
- getValue (String locator) – método que captura o valor de um elemento informado como parâmetro.
- IsEditable (String locator) – método que verifica se um elemento informado como parâmetro é editável.
- isElementPresent (String locator) – método que verifica a existência de um elemento informado como parâmetro.
- getSelectedLabel (String locator) – método que captura o rótulo da opção selecionada em um *drop-down* "locator", informado como parâmetro.
- getSelectedValue (String locator) – método que captura o valor da opção selecionada em um *drop-down* "locator", informado como parâmetro.
- isSomethingSelected (String locator) – método que verifica se há opção selecionada em um *drop-down* "locator", informado como parâmetro.
- IsChecked (String locator) – método que verifica se um *checkbox* está checado.
- getAlert () – método que captura a mensagem de um alerta javascript gerado em uma ação previa no navegador.

2.1.4 *Ant*

O *Apache Ant* (APACHE, 2011a), ou simplesmente *Ant*, é uma ferramenta escrita em Java (JAVA, 2011) e usada para a automatização de builds e tarefas. Para desenvolvedores veteranos, o Ant é uma espécie de make, porém, mais simples de se configurar e usar.

O uso do Ant é justificado devido à quantidade de tarefas que devem ser executadas antes que uma aplicação esteja pronta para instalação ou distribuição final. Entre estas tarefas estão a compilação de classes Java, a criação ou exclusão de diretórios, empacotamento de arquivos, execução de programas externos, etc. Quando se está usando um ambiente integrado de desenvolvimento (*IDE*), este se encarrega de realizar estas tarefas, uma vez que muitos deles já trazem o *Ant* integrado. No entanto, o entendimento do Ant, e suas principais tarefas, é muito importante para o desenvolvedor Java que deseja entender como estas operações funcionam internamente a um IDE.

O *Ant* adiciona portabilidade na automação de *builds* e se propõe a fazer muito mais que isso. Escrito em XML, fica fácil de aprender, e integrar-se com as principais IDEs do mercado. Além do dito anteriormente, Ant possui outras funcionalidades, como por exemplo:

- Manipulação de arquivos e diretórios;
- Substituição de palavras em arquivos;
- Envio de e-mail;
- Conexão com banco de dados;
- Documentação;
- E muitas outras tarefas.

2.1.4.1 Buildfiles

O Ant trabalha com arquivos *XML* (XML, 2011) chamados de *buildfiles*, eles são interpretados pelo *ANT*, para que ele possa executar as tarefas que estão descritas nesses arquivos. O buildfile é um arquivo XML geralmente chamado de build.xml, este arquivo está normalmente organizado desta maneira:

```
<project>
        <propety1/>
        <propety2/>
        <target1>
            <task/>
        </target1>
        <target2>
            <task/>
        </target>
</project>
```

Um *project* é a tag raiz do build.xml, ele representa todo o projeto e só pode existir um por *buildfile*.

Um *target* é uma coleção de tarefas (*tasks*)que desejamos aplicar em determinado momento e encadeando junto com outras tarefas.

Um *task* é uma tarefa que desejamos que seja feita dentro do *target*. O *Ant* já disponibiliza tarefas prontas, porém é possível criar novas.

Um *propety* é um parâmetro em forma de nome-valor necessário para configurar nossa aplicação. Um exemplo da criação de uma *property* seria:
<property name="build" value="bin" />

Para acessar o valor da *property* que foi criada, podemos usar o operador "\$" seguido do nome da propiedade. Assim imprimir o valor da *propety* exemplificada anteriormente poderíamos usar o código abaixo:
<echo>Compilando classer para o diretório: \$build</echo>

2.2 Trabalhos Relacionados

No intuito de simplificar o processo de geração automática de scripts de testes já foram elaborados alguns trabalhos aplicando metodologias e perspectivas diferentes. Esta seção apresenta alguns trabalhos relacionados.

2.2.1 UCT Testing Tool

O trabalho descrito em (CARNIELLO; JINO; CHAIM, 2005) apresenta uma ferramenta de cobertura de testes chamada UCT - *Use Case Tester* - foi desenvolvida para suportar novos critérios de teste. Esta ferramenta tem três funcionalidades principais:

Determinação dos requisitos de teste: a partir de uma análise dos arquivos que descrevem os casos de uso de um sistema, UCT mapeia cada caso de uso para uma representação interna composta de duas listas: uma que contêm as inclusões entre o caso de uso atual e os outros casos de uso; e a outra contêm as extensões. É baseado nestas listas que o testador define os critérios de teste.

Simulação do caso de teste: consiste em percorrer o grafo que representa o caso de teste e determinar quais inclusões e exceções são exercitadas. Assim, a identificação dos requisitos implicados por um conjunto específico de testes permite verificar o comportamento do caso de uso pela análise da sequência de ações simuladas.

Análise dos resultados: uma vez que a sequência de inclusões e extensões exercitada durante o caso de uso simulado é gerada, esta sequência é comparada com os requisitos de simulação dos critérios aplicados aos casos de teste em que a simulação foi executada.

2.2.2 SPACES

O trabalho descrito em (BARBOSA, 2005) apresenta uma ferramenta que tem como objetivo a geração de artefatos de teste para componentes de software a partir do conjunto de artefatos UML e restrições OCL (*Object Constraint Language*) que constituem a especificação dos componentes a serem testados, deste modo, auxiliando o fornecedor a maximizar a qualidade dos componentes disponibilizados. Esta ferramenta tem como funcionalidades principais:

1. Integração com ferramentas de modelagem UML (UML, 2011), desde que as mesmas suportem o formato XMI (XMI, 2011), padrão da OMG (OMG, 2011) para modelos UML em notação XML (XML, 2011).

2. Arquitetura adaptável para diferentes linguagens de restrição (OCL (OCL, 2011), Object-Z (OBJECT-Z, 2011), etc.), assim como a geração de códigos de teste para diferentes linguagens de programação e/ou plataformas (Windows, Linux e Mac).

3. Persistência de sessões de trabalho, permite ao usuário que o mesmo possa interromper e, posteriormente, dar continuidade ao trabalho em qualquer etapa do andamento do projeto.

2.2.3 TDE UML Editor

O trabalho descrito em (VANZIN; MARTINS; FILHO, 2006) apresenta uma ferramenta que permite construções automáticas a partir de modelos UML previamente existentes. Ele é usado para modelar suítes de testes abstratas usando diagramas UML e gerar suítes de testes executáveis a partir deles. Esta ferramenta tem como funcionalidades principais:

- Editor de Diagramas: permite a criação de diagramas UML com auxilio de uma interface rica.
- Validador de Modelos: responsável pela validação dos modelos gerados, ele verifica si o modelo gerado é consistente e passível de geração de código.
- Gerador de Teste: lê os modelos construídos e constrói os executáveis de teste a partir deles.

2.2.4 Análise de Trabalhos Relacionados

Como apresentamos, foram analisadas as ferramentas:

- SPACES;
- UCT Testing Tool;
- TDE UML Editor.

A tabela 2 representa uma análise comparativa entre as ferramentas visando os prós e contras de seus usos, enquanto a tabela 1 representa uma análise comparativa das funcionalidades das ferramentas.

Tabela 1: Tabela de Funcionalidades.
[Fonte: elaboração própria]

Funcionalidade \Ferramenta	**UCT Testing Tool**	**SPACES**	**TDE UML Editor**
Criação\Edição UML	Presente	Presente	Presente
Importar UML	Presente	Presente	Presente
Elaboração Manual de testes	Ausente	Presente	Ausente
Edição de caso de teste	Ausente	Textual, Visual	Textual
Selecionar caso de teste	Presente	Presente	Presente
Selecionar dados de teste	Presente	Presente	Presente
Gerar artefatos	Presente	Presente	Presente
Persistência de dados	Ausente	Presente	Presente
Tipo de teste	Simulação	Execução	Execução
Identificação de cenários	Manual	Automática	Automática
Seleção de Cenários	Manual	Manual/Automática	Manual

Tabela 2: Tabela de Prós e Contras.
[Fonte: elaboração própria]

	UCT Testing Tool	**SPACES**	**TDE UML Editor**
Prós	Reuso de artefatos pré-existentes.	Reuso de artefatos pré-existentes Validação automática dos casos de testes referentes aos cenários selecionados. Identificação e seleção automática dos cenários de testes.	Suporte a ambiente de desenvolvimento distribuído. Integrado ao ambiente de desenvolvimento "Eclipse". Suporte a controle de versões utilizando Subversion (APACHE, 2011b).
Contras	A abrangência dos testes gerados é limitada pela modelagem UML do sistema.		

3 Método de Pesquisa

Nas próximas seções são apresentados mais detalhes sobre a metodologia aplicada no desenvolvimento deste trabalho.

3.1 Qualificação do Método de Pesquisa

Este estudo consiste na sua essência de atividades de planejamento e desenvolvimento de um software que possua as características necessárias à solução do problema de pesquisa proposto inicialmente.

3.2 Etapas do Método de Pesquisa

Este projeto foi divido em fases, sendo elas:

1. Estudo de Conceitos Relacionados: consistindo na realização de pesquisa bibliográfica sobre os conceitos de engenharia de software relacionados à prática de testes e no estudo da plataforma Selenium.

2. Análise de Trabalhos Relacionados: consistindo na realização de pesquisa bibliográfica sobre ferramentas de com proposta similares à deste trabalho.

3. Projeto e Implementação: projeto e implementação de uma ferramenta computacional que possua as características necessárias à solução do problema de pesquisa levantado inicialmente.

4. Definição e Aplicação da Ferramenta: escolha e utilização de uma aplicação como estudo de caso, de forma que o estudo de caso possa contribuir na concepção do projeto, fornecendo um feedback realista para o trabalho.

3.2.1 Projeto e Implementação

Para o desenvolvimento deste trabalho, foi aplicado como padrão de desenvolvimento um modelo iterativo, auto incremental (RUP, 2011), dividido em quatro iterações:

1. Mapeamento: iteração responsável pela definição da estrutura de mapeamento de linguagem natural em blocos de comandos Selenium;
2. Descrição de Caso de uso: iteração responsável por usar os produtos gerados pelo módulo anterior para montar as descrições de um caso de uso;
3. Compilação e Empacotamento: iteração responsável por usar os produtos gerados pelo módulo anterior para gerar os scripts de teste em Java, compilá-los em empacotá-los em executáveis Java;
4. Execução e Análise de Resultados: iteração responsável por usar os produtos gerados pelo módulo anterior para realização do teste.

Para o implementação deste trabalho foi decidido por adotar alguns padrões de projeto (GAMMA RICHARD HELM, 1994):

Singleton Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto. Ele é importante para a ferramenta proposta, pois garante corretude das entidades manipuladas, elimina problemas de concorrência quanto à compilação e execução e controla de modo mais rígido de acesso ao banco de dados nos módulos de gerenciamento de projetos, compilação, execução e utilitário de conexão com banco de dados respectivamente.

Façade Este padrão prove uma interface unificada para um conjunto de interfaces em um subsistema. Ele define uma interface de mais alto nível que torna o subsistema mais fácil de usar. Ele é importante para a ferramenta proposta, por facilitar o reuso e a padronização do código, além de manter as dependências de cada camada internas as mesmas, facilitando a manutenção.

3.3 Limitações da Pesquisa

A fim de alocar o projeto dentro do cronograma estipulado, o mesmo teve seu escopo reduzido em relação aos trabalhos relacionados. Segue uma relação das limitações da ferramenta proposta:

1. Diferentemente dos trabalhos relacionados, nossa proposta não gera os artefatos de testes a partir da especificação UML do sistema a ser testado.

2. A ferramenta proposta abrange apenas testes em sistemas web.

3. Para uso da ferramenta é necessário antes a alimentação da base de dados referente ao conjunto de mapeamentos.

4. Em seu estado atual, a descrição de casos de uso não suporta fluxos alternativos ou de exceção, apenas o fluxo principal.

Além destas limitações, em função do tempo disponível, a ferramenta não pode ser utilizada e avaliada no contexto de um projeto real.

4 Resultados

Neste capítulo, são apresentados os artefatos gerados durante o desenvolvimento da ferramenta SeleniumGenTool, assim como a discussão da mesma.

4.1 Visão de Requisitos

A partir da análise das ferramentas similares, levantou-se um conjunto de requisitos que podem ser vistos na Tabela 3.

Tabela 3: Tabela de Requisitos.
[Fonte: elaboração própria]

Nome	**Descrição**
Gerenciar Projetos	Permite ao testador criar e excluir projetos. Os projetos contêm as estruturas necessárias para a manipulação dos testes.
Gerenciar Mapeamentos	Permite ao testador criar e excluir relações entre linguagem natural e comandos Selenium. É a partir dessas relações que serão gerados os scripts de teste.
Gerenciar Caso de Uso	Permite ao testador criar e editar os casos de uso que descrevem o sistema a ser testado. Contêm as estruturas necessárias para a manipulação dos dados que descrevem as funcionalidades do sistema a ser testado.
Gerenciar Cenários	Permite ao usuário escolher, dentre os cenários identificados, quais serão aqueles explorados para testes, bem como prover os dados necessários para os mesmos.
Gerenciar Scripts de Teste	Permite ao usuário gerar, executar e analisar os resultados dos scripts de teste para os cenários determinados no ambiente desejado.

4.2 Visão de Caso de Uso

Nesta seção é apresentado um diagrama de caso de uso da ferramenta proposta. A Figura 5 simboliza o diagrama citado e o mesmo apresenta um conjunto de cenários que cobrem os principais requisitos levantados para a ferramenta.

Uma breve descrição de alguns casos de uso é apresentada a seguir:

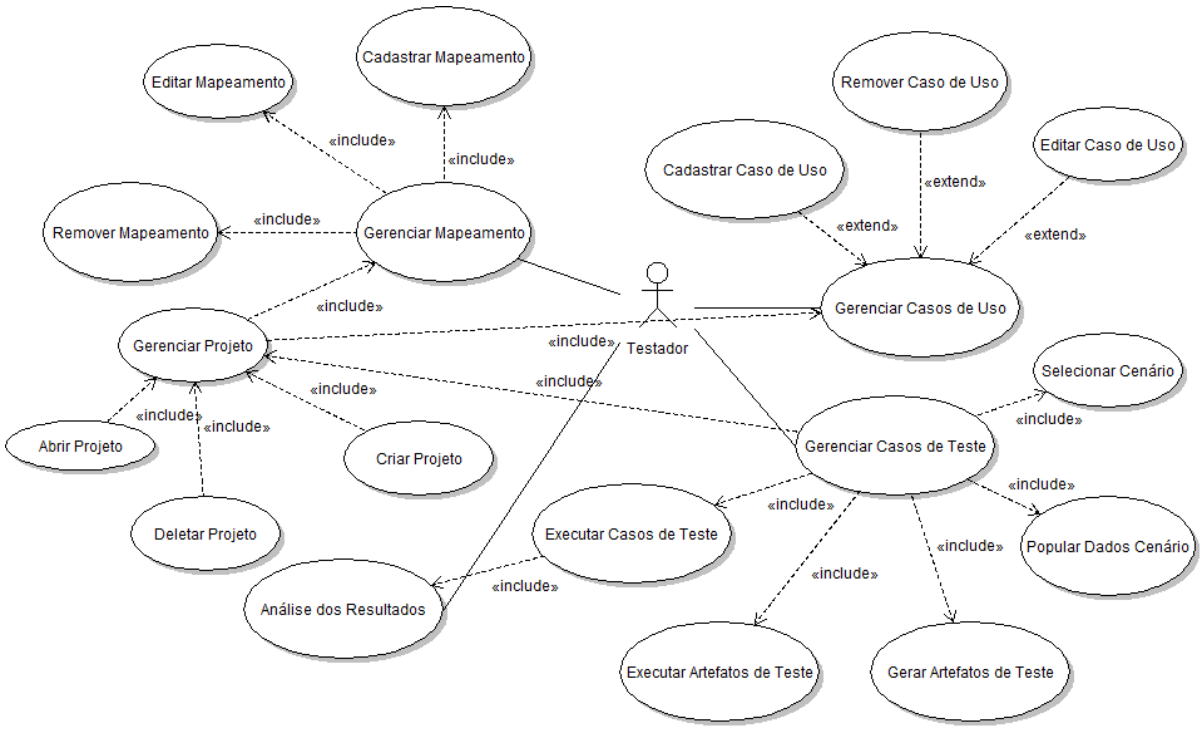

Figura 4: Casos de Uso do SeleniumGenTool.

UC001 – Gerenciar Projeto

- Pré-condições: nenhuma.
- Descrição: o usuário deseja elaborar um novo projeto de testes. Para isso ele solicita a criação de um novo projeto \abertura de um projeto já existente através do botão localizado na interface inicial da ferramenta ou na opção referente oferecida no menu arquivo. Neste módulo o usuário informa informações básicas sobre o sistema a ser testado tais quais a url do mesmo e confirma a criação do mesmo. Ao confirmar a criação, a ferramenta retorna o registro referente ao projeto criado \selecionado no banco de dados.
- Pós-condições: o projeto selecionado \criado é definido como o projeto corrente na execução da ferramenta.

UC002 – Gerenciar Mapeamento

- Pré-condições: nenhuma.
- Descrição: o usuário deseja criar \editar uma relação linguagem natural $\leftrightarrow$ comando Selenium. Para isso ele solicita a criação de um novo mapeamento através do botão localizado na interface inicial da ferramenta ou na opção referente no menu arquivo. Neste módulo o usuário informa as informações

referentes à descrição textual que poderá ser usada na composição de um caso de uso, sua classificação e seu bloco de comandos Selenium equivalentes. Ao confirmar a criação, a ferramenta adiciona o registro referente ao novo mapeamento ao banco de dados.

- Pós-condições: um mapeamento é gerado \editado.

UC003 – Gerenciar Caso de Uso

- Pré-condições: existência previa de um projeto ([UC001]); existência previa de um ou mais mapeamentos ([UC002]).
- Descrição: o usuário deseja incluir \editar um caso de uso do sistema a ser testado. Para isso ele solicita a criação de um novo caso de uso através do botão localizado na interface inicial da ferramenta ou na opção referente no menu arquivo. Neste módulo o usuário informa informações referentes à identificação do caso de uso tais quais nome (opcional) e descrição (opcional) e referentes à descrição do mesmo, fazendo uso dos mapeamentos cadastrados previamente. ao confirma a criação \edição, a ferramenta adiciona o registro referente ao caso de uso cadastrado associado ao projeto corrente.
- Pós-condições: um caso de uso gerado \editado é no projeto.

UC004 – Gerenciar Cenários

- Pré-condições: existência previa de um projeto ([UC001]); existência previa de um caso de uso ([UC003]).
- Descrição: o usuário deseja elaborar os cenários derivados dos casos de uso existentes no projeto. Para isso ele solicita a geração de cenários do botão localizado na interface inicial da ferramenta ou na opção referente no menu arquivo. Neste módulo o usuário informa os valores referentes aos parâmetros informado na etapa de mapeamento além de informar qual será o navegador no qual o teste será executado. Ao confirmar a criação, a ferramenta adiciona o registro referente ao(s) novo(s) cenário(s), derivado(s) de um caso de uso referido, ao banco de dados.
- Pós-condições: um ou mais novos cenários gerados são preparados no projeto.

UC005 – Gerar Scripts de Teste

- Pré-condições: existência previa de um projeto ([UC001]); existência previa de um caso de teste ([UC004]).

- Descrição: o usuário deseja gerar os artefatos de teste. Para isso ele deve selecionar a opção referente no menu testes ou acionar o botão referente na interface de cenários e optar dentre os casos de teste existentes, após isso caos de teste selecionado será validado. Caso não haja dados inválidos ou faltantes em relação aos passos relacionados no caso de teste selecionado será convertido em código Java, caso não o processo será interrompido. Ao confirmar a criação, a ferramenta adiciona o registro referente ao novo cenário, derivado de um caso de uso referido, ao banco de dados.
- Pós-condições: um novo script de teste é gerado.

UC006 – Gerar Artefatos de Teste

- Pré-condições: existência previa de um projeto ([UC001]); existência previa de um script de teste ([UC005]).
- Descrição: o usuário deseja gerar os artefatos de teste. Para isso ele deve selecionar a opção referente no menu testes ou acionar o botão referente na interface de cenário e optar dentre os casos de teste existentes. O usuário deverá informar um diretório válido para a exportação dos artefatos. Ao confirmar a criação, a ferramenta exportará o código Java referente ao script de teste selecionado, o compilará e o empacotará em um executável Java. O relatório da execução dessas atividades será informado ao usuário.
- Pós-condições: um novo conjunto de artefatos de teste é gerado.

UC007 – Executar Artefatos de Teste

- Pré-condições: existência previa de um projeto ([UC001]); existência previa de um artefato de teste ([UC005]).
- Descrição: o usuário deseja executar os artefatos de teste. Para isso ele deve selecionar a opção referente no menu testes ou acionar o botão referente na interface de execução de testes e informar os artefatos de teste que deseja executar. Ao confirmar o início da execução, a ferramenta executa cada artefato de teste informado e informa a saída de cada execução.
- Pós-condições: um relatório de execução de artefatos de teste.

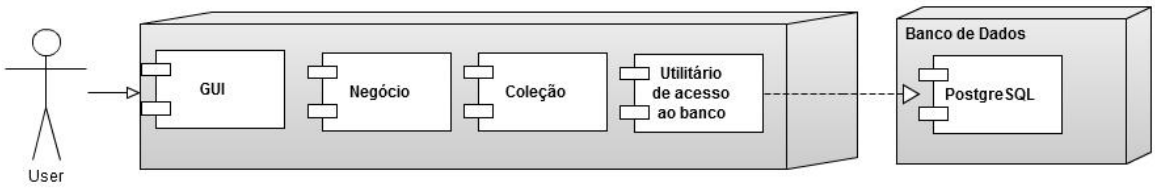

Figura 5: Arquitetura, Visão Geral.

4.3 Visão Geral da Arquitetura

A arquitetura da ferramenta consiste, basicamente, de uma arquitetura em camadas (GAMMA RICHARD HELM, 1994), onde temos:

- GUI: camada\subsistema de apresentação e comunicação com o usuário, ou seja, camada de interface gráfica (telas) de comunicação com o usuário do sistema em questão.
- Negócio: camada de negócio do sistema, onde serão implementadas as regras do sistema e realizadas as validações de negócio existentes no sistema em questão.
- Coleção: esta camada recebe as requisições da camada de negócios e seus métodos executam essas requisições em um banco de dados. Caso se opte por outro banco de dados, basta alterar apenas as classes da camada de dados, e as demais camadas não são afetadas por essa alteração.
- Utilitário de Acesso ao Banco: gerencia as conexões de acesso ao banco de dados.

4.4 Visão Lógica da Arquitetura

A Figura 6 representa uma visão de componentes da ferramenta. Nela ficam claros os padrões de projeto aplicados no desenvolvimento da ferramenta (*Façade* e *Singleton*) (GAMMA RICHARD HELM, 1994).

A visão lógica da arquitetura da ferramenta, basicamente, consiste em:

Gerenciador de Projetos: interface gráfica com usuário para gestão de projetos.

Gerenciador de Mapeamentos: interface gráfica com usuário para gestão de mapeamentos.

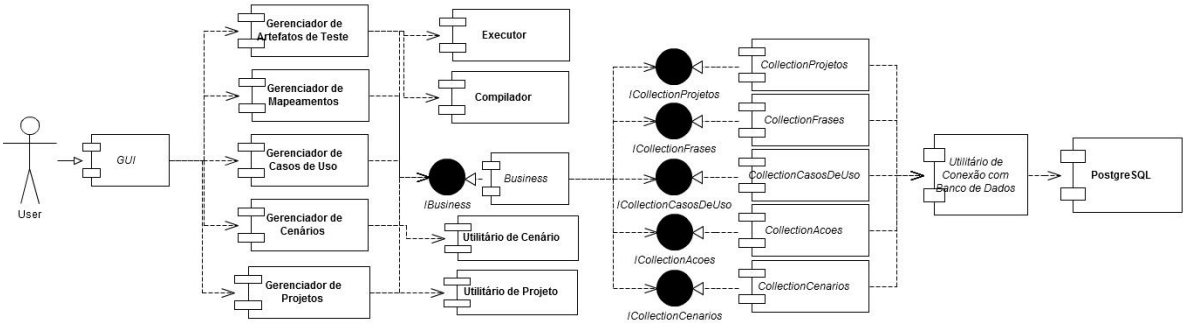

Figura 6: Arquitetura, Visão Lógica.

Gerenciador de Casos de Uso: interface gráfica com usuário para gestão de casos de uso.

Gerenciador de Cenários: interface gráfica com usuário para gestão de cenários.

Gerenciador de Scripts de Teste: interface gráfica com usuário para gestão de casos de teste.

Utilitário de Cenários: componente responsável pela manipulação dos dados referentes à cenários com o intuito de gerar os scripts de teste.

Utilitário de Projetos: componente responsável pela manipulação dos dados referentes ao conjunto de entidades que compõe o projeto.

Compilador: componente responsável pela compilação e empacotamento dos artefatos de teste.

Executor: componente responsável pela execução dos artefatos de teste.

IBusiness: interface de acesso à camada de negócios do projeto.

Business: camada que implementa as regras de negócio do projeto.

ICollectionProjetos: interface de acesso à camada de repositório de projetos da ferramenta.

ICollectionFrases: interface de acesso à camada de repositório de frases da ferramenta.

ICollectionCasosDeUso: interface de acesso à camada de repositório de casos de uso da ferramenta.

ICollectionAcoes: interface de acesso à camada de repositório de ações da ferramenta.

ICollectionCenarios: interface de acesso à camada de repositório de cenários da ferramenta.

CollectionProjetos: camada que implementa a coleção de dados referente a projetos na ferramenta.

CollectionFrases: camada que implementa a coleção de dados referente às frases na ferramenta.

CollectionCasosDeUso: camada que implementa a coleção de dados referente aos casos de uso na ferramenta.

CollectionAcoes: camada que implementa a coleção de dados referente às ações na ferramenta.

CollectionCenarios: camada que implementa a coleção de dados referente aos cenários na ferramenta.

Utilitário de Conexão com Banco de Dados: utilitário de controle de acesso ao banco de dados da ferramenta.

Referente ao modelo Entidade-Relacional, sua representação pode ser vista na Figura 7. Nela pode ser visto que para atingir o objetivo proposto, a ferramenta faz uso das seguintes entidades:

Projeto: responsável pela definição do escopo de uma atividade de testes.

Frase: responsável pela definição do mapeamento entre linguagem natural e comandos Selenium.

Caso de Uso: responsável pela definição dos casos de uso a serem testados.

Ação: responsável pela definição da relação de frases associadas à casos de uso.

Cenário: responsável pela definição da situação a ser testada.

Nas seções a seguir, segue uma explicação mais detalhada sobre os principais módulos da ferramenta.

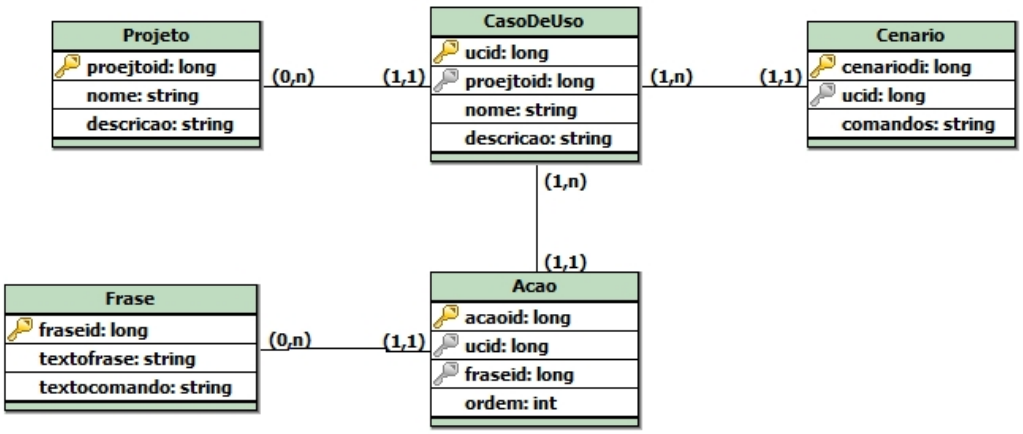

Figura 7: Modelo ER.

4.5 Ferramenta SeleniumGenTool

Com o intuito de prover uma interface de uso mais intuitivo, a interface da ferramenta foi divida em módulos e a navegação entre os módulos se dá através de abas conforme pode ser visto na Figurañeffig:navegacao. Foi implementada uma aba para cada módulo. A seguir descreveremos em detalhes cada aba.

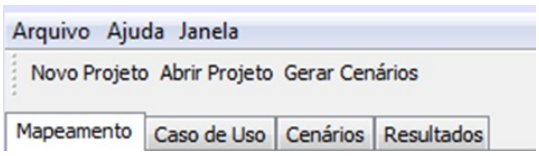

Figura 8: SeleniumGenTool UI – Navegação.

4.5.1 Gerenciamento de Projeto

O objetivo desta etapa é criar \abrir projetos de testes. As Figuras 9 e 10 são referente às telas dos módulo em questão.

Como pode ser visto Figuras 9, esta interface é composta, basicamente, por dois elementos:

- O primeiro é referente a um campo de texto que deve ser utilizado para informar o nome dado ao projeto.
- O segundo refere-se a uma área de texto que deve ser utilizada para informar uma descrição do projeto.

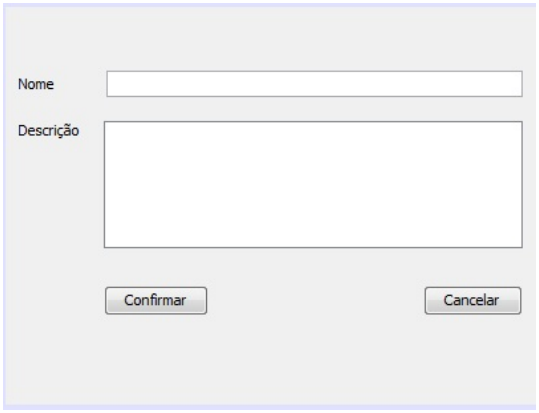

Figura 9: SeleniumGenTool UI – Novo Projeto.

Ambos os campos são opcionais.

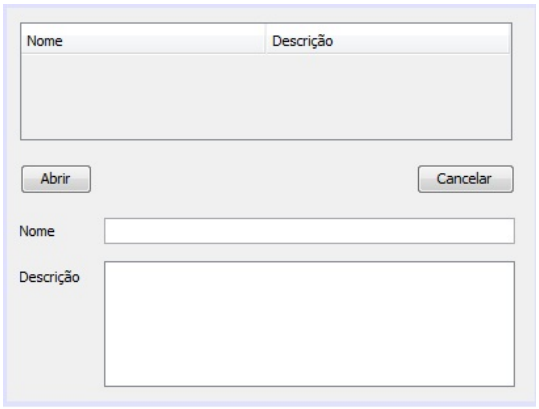

Figura 10: SeleniumGenTool UI – Abrir Projeto.

Como pode ser visto Figuras 10, esta interface é composta, basicamente, por dois elementos:

- O primeiro é referente a uma tabela que exibe a coleção de projetos já cadastrados. Através dela é possível selecionar o projeto desejado.
- O segundo é referente a uma área destinada à identificação do projeto. Ela é subdividida em dois elementos:
  - O primeiro é referente a um campo de texto que informará o nome projeto do projeto selecionado.
  - O segundo refere-se a uma área de texto que informará a descrição do projeto selecionado.

4.5.2 Gerenciado de Mapeamento Linguagem Natural ↔ Selenuim

O objetivo desta etapa é estabelecer uma relação direta entre linguagem natural e um bloco de comandos Selenium. A Figura 11 é referente à tela do módulo em questão.

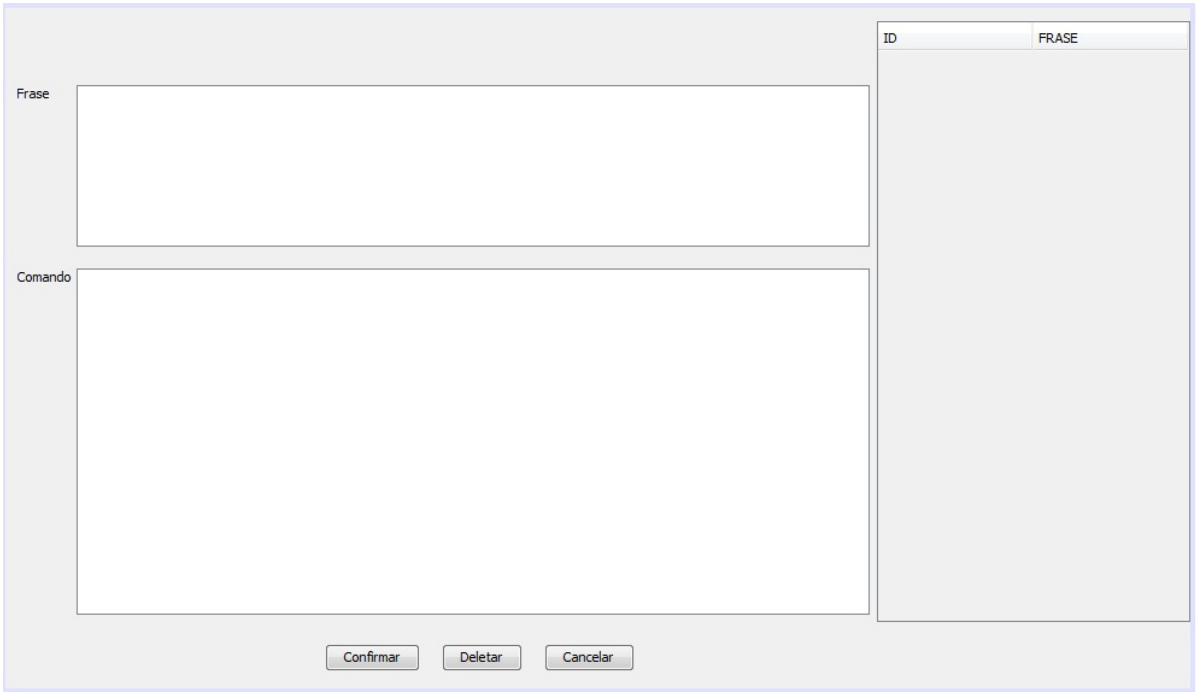

Figura 11: SeleniumGenTool UI – Gerenciador de Mapeamento.

Como pode ser visto na figura, esta interface é composta, basicamente, por dois elementos:

- O primeiro, localizado na parte direita da imagem, é referente a uma tabela que exibe a coleção de mapeamentos já cadastrados. Através dela é possível selecionar mapeamentos para edição ou deleção.
- O segundo é subdividido em outros itens:
  - O primeiro refere-se a uma área de texto que deve ser utilizada para descrever a frase que representará o mapeamento, lembrando que será baseado nessa descrição que o usuário fará suas escolhas para montar as descrições dos casos de uso na interface de caso de uso.
  - O segundo, no centro da imagem, refere-se a uma área de texto que deve ser utilizada para informar o bloco de comandos Java referentes à ação desejada,

lembrando que para interação do script de teste com o ambiente web deve-se fazer o uso de comandos Selenium. Foi definido, como padrão para reconhecimento de variáveis, que para passagem de parâmetros será necessário o uso do marcador '#' no início e fim do identificador do parâmetro. Será está marcação o que definirá a necessidade de informação de valores no módulo de gerenciamento de cenários.

4.5.3 Gerenciador de Casos de Uso

O objetivo desta etapa é fazer uso das frases mapeadas em blocos de comando Selenium na etapa anterior para, assim, descrever o sistema a ser testado. A Figura 12 é referente à tela do módulo em questão.

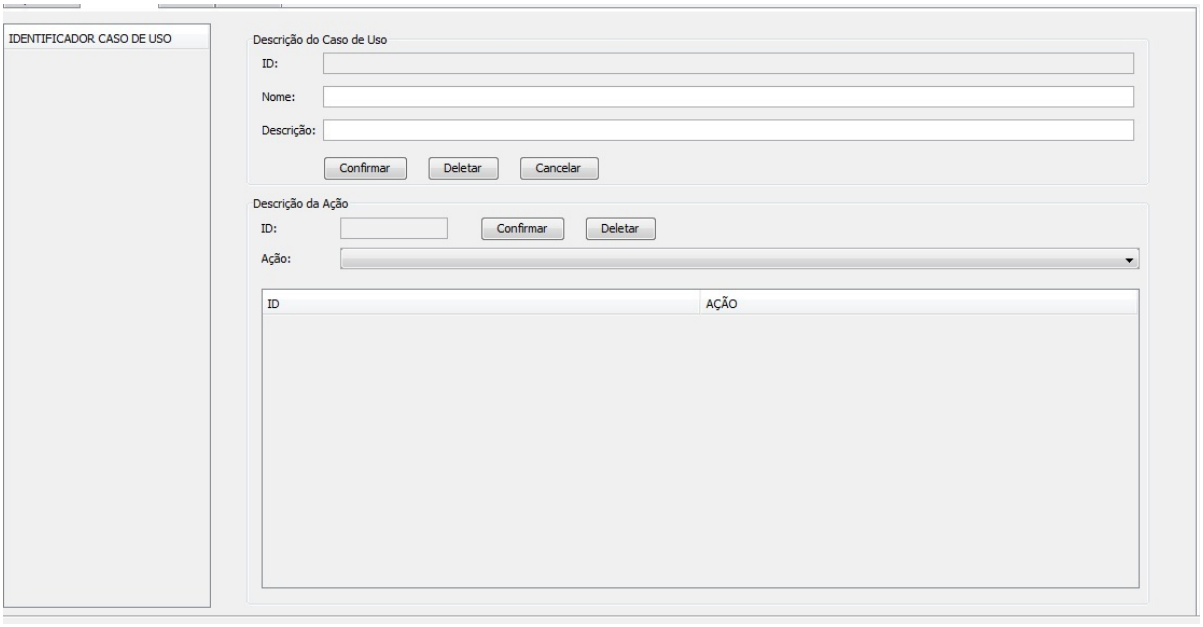

Figura 12: SeleniumGenTool UI – Gerenciador de Casos de Uso.

Como pode ser visto na figura, esta interface é composta, basicamente, por três elementos:

- O primeiro é referente a uma tabela que exibe a coleção de casos de uso já cadastrados e associados ao projeto corrente. Através dela é possível selecionar casos de uso para edição ou deleção.
- O segundo é referente a uma área destinada à identificação do caso de uso corrente. Ela é subdividida em três elementos:

  - O primeiro trata-se de um campo de texto destinado a exibir o identificador único de cada caso de uso. Este campo não é editável.
  - O segundo trata-se de um campo de texto destinado a receber um nome, dado pelo usuário, ao caso de uso corrente. Este campo é opcional.
  - O terceiro trata-se de um campo de texto destinado a receber uma breve descrição do caso de uso. Este campo é opcional.

- O terceiro é subdividido em outros itens:
  - O primeiro refere-se a uma tabela que exibe a coleção de ações já cadastradas em um caso de uso. Através dela é possível selecionar ações para deleção.
  - O segundo trata-se de um campo de texto destinado a exibir o identificador único de cada ação. Este campo não é editável.
  - O terceiro refere-se a um seletor que exibe uma lista de frases associadas a blocos de comando (mapeamentos) para serem adicionados como ação a um caso de uso.

4.5.4 Gerenciador de Cenários

O objetivo desta etapa é gerir os possíveis cenários de teste derivados dos casos de uso estabelecidos na etapa anterior. Para cada relação cenário $\leftrightarrow$ valores informados é gerado um script de testes, em Java, referente. A Figura 13 é referente à tela do módulo em questão.

Como pode ser visto na figura, esta interface é composta, basicamente, por quatro elementos:

- O primeiro é referente a um seletor, ele é referente ao filtro que será aplicado na consulta que informará a coleção de cenários derivados do caso de uso selecionado.
- O segundo é referente a um botão responsável pela ação de geração de executáveis derivados do script de testes associado ao cenário corrente.
- O terceiro é referente a uma tabela que exibe a coleção de cenários já cadastrados derivado do caso de uso corrente. Através dela é possível selecionar cenários para edição.
- O quarto é referente a uma área destinada à passagem de valores para o cenário corrente. Ela é subdividida em dois elementos:

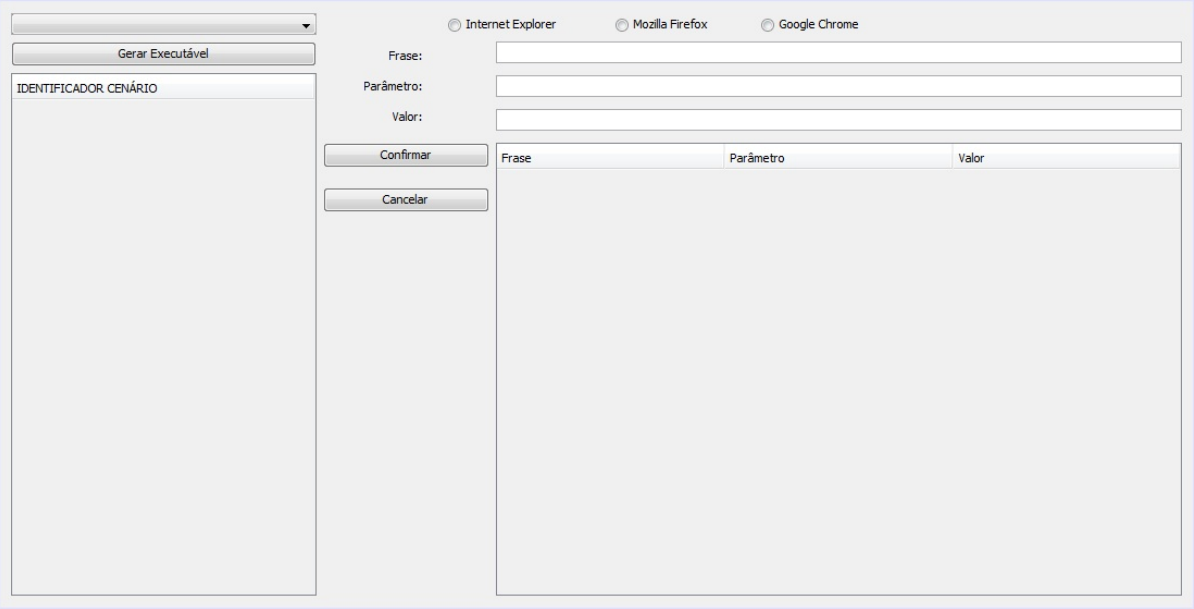

Figura 13: SeleniumGenTool UI – Gerenciador de Cenários.

- O primeiro refere-se a uma tabela que exibe a coleção de parâmetros associados ao cenário corrente. Através dela é possível selecionar parâmetros para passagem e valores.
- O segundo é subdividido em outros itens:
  - O primeiro refere-se a um conjunto de botões relativos aos possíveis navegadores alvo do teste. Através dele é possível selecionar qual dos navegadores será o ambiente de testes.
  - O segundo trata-se de um campo de texto destinado a exibir a frase referente à ação no qual o parâmetro se encontra. Este campo não é editável.
  - O terceiro refere-se a um campo de texto destinado a exibir o parâmetro corrente da frase ao qual se deseja informar um valor. Este campo não é editável.
  - O quarto refere-se a um campo de texto destinado a receber o valor referente ao parâmetro selecionado.

4.5.5 Geração de Artefatos de Teste

O objetivo desta etapa é a geração dos scripts e executáveis de teste derivados dos cenários gerados na etapa anterior. O andamento da execução e compilação do script é informado ao usuário. A Figura 14 é referente à tela do módulo em questão.

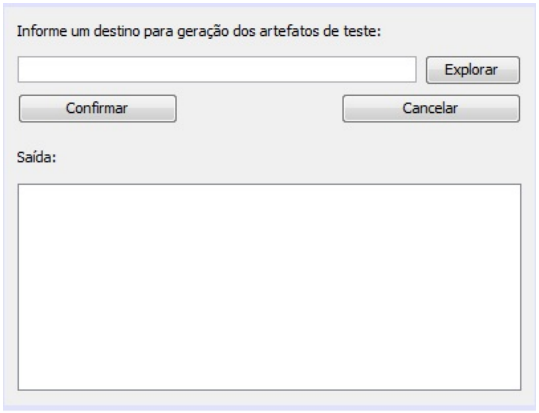

Figura 14: SeleniumGenTool UI – Geração de Artefatos.

Como pode ser visto na figura, esta interface é composta, basicamente, por dois elementos:

- O primeiro é referente a uma área destinada à informação do destino no qual os artefatos de teste serão gerados. Ela pode ser feita de dois diferentes modos:
  - O primeiro se da através do preenchimento direto do campo de texto referente ao caminho para a pasta destino.
  - O segundo se da através da seleção de uma pasta através do explorador de pastas associado ao botão "Explorar´´.
- O segundo é referente a uma área de texto na qual será exibida o andamento da execução da geração dos artefatos.

4.5.6 Execução e Análise dos Resultados

O objetivo desta etapa é a execução e analise dos resultados desta execução referente aos artefatos teste gerados na etapa anterior. A Figura 15 é referente à tela do módulo em questão.

Como pode ser visto na figura, esta interface é composta, basicamente, por três elementos:

- O primeiro é referente a uma área destinada à informação do destino no qual os artefatos de teste gerados se encontram. Ela pode ser feita de dois diferentes modos:
  - O primeiro se da através do preenchimento direto do campo de texto referente ao caminho da pasta destino.

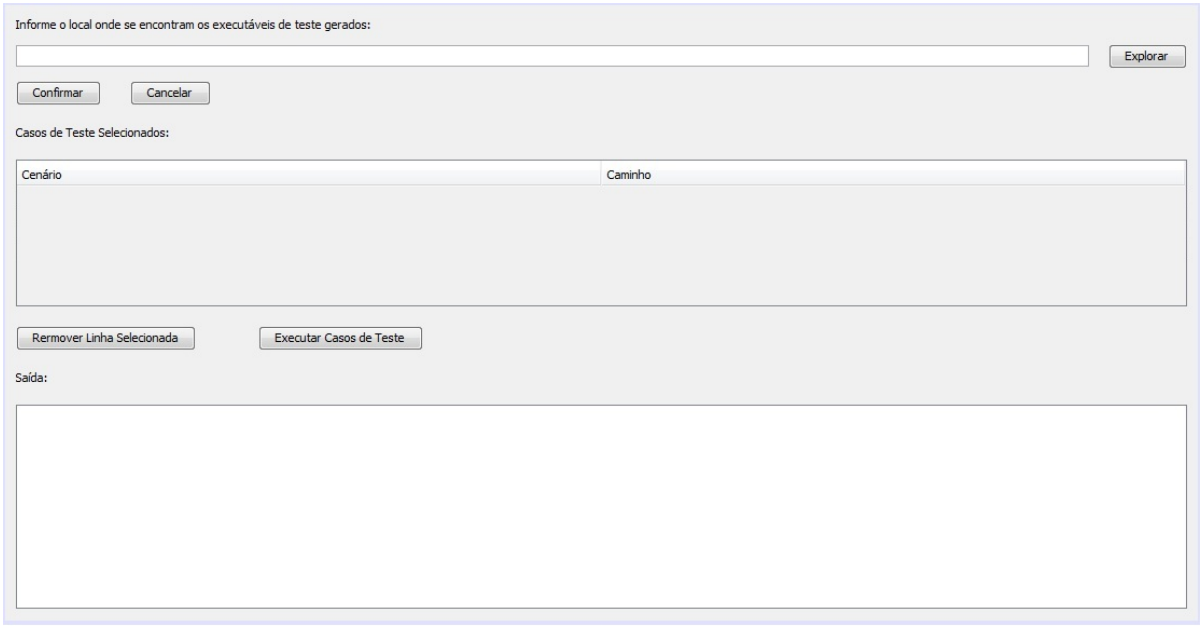

Figura 15: SeleniumGenTool UI – Execução de Artefatos e Análise de Resultados.

  - O segundo se da através da seleção de um arquivo através do explorador de pastas associado ao botão "Explorar".

- O segundo é referente a uma tabela listando todos os artefatos selecionados e seus respectivos caminhos.
- O terceiro é referente a uma área de texto na qual serão exibidos os relatórios de execução dos artefatos de teste.

4.6 Aplicação da Ferramenta

Para demonstração da eficiência desta ferramenta, esta seção apresenta o processo de um exemplo de uso da mesma.

4.6.1 Definição do Projeto

1. O usuário inicializa a ferramenta e aciona o botão referente à "Novo Projeto" na barra de atalhos da ferramenta.
2. Na janela que se abre, o usuário informa um nome para o projeto (Teste de Busca Google) e uma descrição para o mesmo (teste no sistema de busca do Google).

3. O usuário confirma o cadastro do novo projeto acionando o botão referente à "Confirmar".

O fim desta etapa representa a criação de um projeto de testes.

4.6.2 Mapeamento

Esta seção representa a criação(cadastramento) de um mapeamento entre linguagem natural e blocos de comando Selenium.

1. O usuário seleciona a aba "Mapeamento" referente a mapeamentos.
2. O usuário informa o valor "abre a pagina inicial do google" para o campo frase.
3. O usuário informa o valor "driver.get("#url#");" para o campo comando.
4. O usuário confirma o cadastro do novo projeto acionando o botão referente à "Confirmar".
5. O usuário informa o valor "procura pelo campo de informação da palavra a ser pesquisada" para o campo frase.
6. O usuário informa o valor "WebElement element = driver.findElement(By.name("q"));" para o campo comando.
7. O usuário confirma o cadastro do novo projeto acionando o botão referente à "Confirmar".
8. O usuário informa o valor "informa o que deseja pesquisar e aciona o evento de busca" para o campo frase.
9. O usuário informa o valor "element.sendKeys(" #palavra# ");
   element.submit();
   (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
   public Boolean apply(WebDriver d) {
   return d.getTitle().toLowerCase().startsWith(" #palavra# ");
   } });" para o campo comando.

10. O usuário confirma o cadastro do novo projeto acionando o botão referente à "Confirmar".
11. O usuário informa o valor "encerra a pesquisa" para o campo frase.
12. O usuário informa o valor "driver.quit();" para o campo comando.
13. O usuário confirma o cadastro do novo projeto acionando o botão referente à "Confirmar".

O fim desta etapa representa a adição de um ou mais mapeamentos ao cadastro de mapeamentos de linguagem natural para comandos Selenium.

4.6.3 Escrevendo Caso de Uso

Esta seção representa a criação(cadastramento) de uma descrição de casos de uso utilizando-se dos mapeamentos previamente cadastrados.

1. O usuário seleciona a aba "Caso de Uso" referente a casos de uso.
2. O usuário informa o valor "teste de busca" para o campo "Nome";
3. O usuário informa o valor "acionando mecanismo de busca do google" para o campo "Descrição";
4. O usuário confirma o cadastro do novo caso de uso acionando o botão referente à "Confirmar".
5. O usuário seleciona o caso de uso gerado na tabela "IDENTIFICADOR CASO DE USO".
6. O usuário seleciona a ação "abre a pagina inicial do google" na lista de ações.
7. O usuário confirma a adição da ação acionando o botão referente à "Confirmar".
8. O usuário seleciona a ação "procura pelo campo de informação da palavra a ser pesquisada" na lista de ações.

9. O usuário confirma a adição da ação acionando o botão referente à "Confirmar".
10. O usuário seleciona a ação "informa o que deseja pesquisar e aciona o evento de busca" na lista de ações.
11. O usuário confirma a adição da ação acionando o botão referente à "Confirmar".
12. O usuário seleciona a ação "encerra a pesquisa" na lista de ações.
13. O usuário confirma a adição da ação acionando o botão referente à "Confirmar".

O fim desta etapa representa a adição da descrição de um caso de uso ao projeto.

4.6.4 Cenários

1. O usuário aciona o botão referente à "Gerar Cenários" na barra de atalhos da ferramenta. Esta ação resulta na geração de todos os cenários derivados de todos os casos de uso associados ao projeto em execução.
2. O usuário seleciona o caso de uso existente na lista de casos de uso.
3. O usuário seleciona o cenário existente na tabela de cenários.
4. O usuário seleciona o parâmetro "#url#" na tabela de parâmetros.
5. O usuário informa o valor "http://www.google.com" para o parâmetro selecionado.
6. O usuário confirma a passagem de parâmetros acionando o botão referente à "Confirmar".
7. O usuário seleciona o parâmetro "#palavra#" na tabela de parâmetros.
8. O usuário informa o valor "Cheese!" para o parâmetro selecionado.
9. O usuário confirma a passagem de parâmetros acionando o botão referente à "Confirmar".
10. O usuário seleciona o parâmetro "#palavra#" na tabela de parâmetros.

11. O usuário informa o valor “Cheese!” para o parâmetro selecionado.
12. O usuário confirma a passagem de parâmetros acionando o botão referente à “Confirmar”.
13. O usuário seleciona o navegador “Mozilla Firefox”.

O fim desta etapa representa a construção de um cenário derivado de um caso de uso associado ao projeto em execução.

4.6.5 Geração de Artefatos de Teste

1. O usuário aciona o botão referente à “Gerar Executável” na interface de gerenciamento de cenários. Esta ação resulta na abertura de uma janela de gerenciamento de geração de artefatos.
2. O usuário informa o valor “C:\testegoogle>” para o campo “Informe um destino para geração dos artefatos de teste”.
3. O usuário confirma a geração de artefatos acionando o botão referente à “Confirmar”.

O fim desta etapa representa a geração dos artefatos de teste e são eles:

- O script de compilação e construção Ant, encontrado no Apêndice-A.
- O código Java referente ao script de teste Selenium abaixo:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
   public class TestCase_1 {
      public static void main(String[] args)
         WebDriver driver = new FirefoxDriver();
         driver.get(“http://www.google.com”);
```

```
        WebElement element = driver.findElement(By.name("q"));
        element.sendKeys("Cheese!");
         element.submit();
        (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>()
        {
            public Boolean apply(WebDriver d) {
                return d.getTitle().toLowerCase().startsWith("cheese!");
            }
        });
        driver.quit();
    }
}
```

4.6.6 Execução de Artefatos de Teste

1. O usuário seleciona a aba "Resultados" referente à execução e análise dos resultados da execução.

2. O usuário informa o valor "C:\testegoogle\dist\TesteCase_1.jar>" para o campo "Informe o local onde se encontram os executáveis de teste gerados".

3. O usuário confirma a inserção do artefato à lista de execução acionando o botão referente à "Confirmar".

4. O usuário confirma a execução do artefato acionando o botão referente à "Executar Casos de Teste".

O fim desta etapa representa o fim do exemplo.

5 Considerações Finais

A atividade de testes é uma importante etapa do desenvolvimento de software por ajudar a garantir a qualidade do produto desenvolvido; porém, ainda é uma atividade muito custosa. Deve-se a este fato a importância de um bom processo de automação desta atividade.

Existem no mercado algumas ferramentas com esse intuito. Algumas delas foram abordadas neste trabalho, porém este tipo de ferramenta depende da existência de outros artefatos (documento de casos de uso, modelo UML) e isso demanda uma nova atividade, caso a mesma não seja utilizada no modelo de desenvolvimento em questão.

A ferramenta desenvolvida neste trabalho tem como objetivo minimizar a necessidade de artefatos específicos, uma vez que parte da descrição textual de casos de uso que, por sua vez, é um dos poucos artefatos comumente criado pela maioria das empresas de desenvolvimento de software.

Este trabalho atendeu aos objetivos propostos, pois a ferramenta desenvolvida foi capaz de gerar scripts de teste Selenium a partir de texto em linguagem natural. Contribuindo, assim, para diminuição dos custos necessários a atividades de teste em sistemas web. Contudo, a ferramenta pode ser melhorada em vários aspectos, dando margem a trabalhos futuros, alguns dos quais foram identificados e listados na próxima seção.

5.1 Trabalhos Futuros

Como trabalhos futuros, podem-se destacar as seguintes atividades:

1. Adequar a estrutura da ferramenta para que ela possa ser implantada em um sistema web.

2. Ajustar a ferramenta de modo que ela suporte recursos comuns a editores de texto de IDE's, tais quais: auto completar, *syntax highlighting*, etc..

3. Ajustar a ferramenta para que seja possível gerar suítes de teste, composta por vários casos de teste.

4. Lidar com fluxos alternativos e de exceção;

5. Implementar técnicas de seleção em função de cobertura e propósito de testes;

6. Gerenciamento da evolução dos requisitos.

Referências

APACHE. *Apache Ant*. 2011. Website. `http://ant.apache.org/`.

APACHE. *Apache Subversion*. 2011. Website. `http://subversion.apache.org/`.

BARBOSA, D. L. *Um Método Automático de Teste Funcional para a Verificação de Componentes*. Dissertação (Mestrado) — Universidade Federal de Campina Grande, 2005.

CARNIELLO, A.; JINO, M.; CHAIM, M. L. Structural testing with use cases. *Journal of Computer Science & Technology*, 2005.

GAMMA RICHARD HELM, R. J. J. V. E. *Design Patterns : Elements of Reusable Object-Oriented Software*. [S.l.]: Pearson, 1994.

JAVA: Site. 2011. Disponível em: <`http://www.java.com/pt_BR/`>. Acesso em: 20 nov.2011.

MANUAL Selenium: Site. 2011. Disponível em: <`http://seleniumhq.org/docs/`>. Acesso em: 20 nov.2011.

MCGREGOR, J. D.; SYKES, D. A. *A Practical Guide to Testing Object-Oriented Software*. [S.l.]: Addison-Wesley, 2001.

NETO P. S., d. S. F. V.; RESENDE, R. *Automação de Teste de Software*. [S.l.]: Escola Regional de Computação da SBC Ceará - Maranhão - Piaui, Fortaleza., 2007.

OBJECT-Z: Site. 2011. Disponível em: <`http://itee.uq.edu.au/~smith/objectz.html`>. Acesso em: 20 nov.2011.

OCL: Site. 2011. Disponível em: <`http://www.omg.org/spec/OCL/2.2/`>. Acesso em: 20 nov.2011.

OMG: Site. 2011. Disponível em: <`http://www.omg.org/`>. Acesso em: 20 nov.2011.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. [S.l.]: McGraw-Hill, 2005.

RUP: Site. 2011. Disponível em: <`http://www.wthreex.com/rup/`>. Acesso em: 20 nov.2011.

SELENIUM: Site. 2011. Disponível em: <`http://seleniumhq.org/`>. Acesso em: 20 nov.2011.

SOMMERVILLE, I. *Engenharia de Software* . [S.l.]: Addison-Wesley, 2007.

SWEBOK. *Guide to the Software Engineering Body of Knowledge* . [S.l.]: IEEE, 2004.

TORRES, F. B. B. e I. V. *O Teste de Software no Mercado de Trabalho*. v. 2 n. 1. [S.l.], junho 2011. P. 49-52.

UML: Site. 2011. Disponível em: <`http://www.uml.org/`>. Acesso em: 20 nov.2011.

VANZIN, D. D.; MARTINS, I. L.; FILHO, J. B. A. P. Tde uml editor - a success development case of a software extension. In: *Global Software Engineering, 2006. ICGSE '06. International Conference on*. [S.l.: s.n.], 2006. p. 257 –258.

XMI: Site. 2011. Disponível em: <`http://www.omg.org/spec/XMI/`>. Acesso em: 20 nov.2011.

XML: Site. 2011. Disponível em: <`http://www.w3.org/XML/`>. Acesso em: 20 nov.2011.

APÊNDICE A – Script Ant

<?xml version="1.0" encoding="iso-8859-1" ?>

<!– Configuramos como default o nosso target "dist", nele temos um atributo chamado depends, que indica uma dependência em relação a outro alvo. Cada alvo será executado na ordem em que for chamado respeitando a ordem de suas dependências. –>

<project name="TesteAnt" default="dist" basedir="C:\testegoogle\">

<!– propriedades do projeto com os nomes dos diretórios –>

<property name="src" value="C:\testegoogle\" />

<property name="build" value="C:\testegoogle\classes" />

<property name="dist" value="C:\testegoogle\dist" />

<!– target init - target responsável por criar os diretórios classes, doc e dist. –>

<target name="init">

<echo> Criando os diretório classes, doc e dist.</echo>

<mkdir dir="\$build" />

<mkdir dir="\$dist" />

</target>

<!– target compile - target responsável por compilar o projeto separando os arquivos .class no diretório classes. –>

<target name="compile" depends="init">

<echo> Compilando o projeto.</echo>

<javac srcdir="C:\testegoogle\" destdir="\$build"/>

</target>

<!– target dist - target responsável por gerar o .jar executável do projeto no diretório dist. –>

<target name="dist" depends="compile">

<echo> Gerando o .jar do projeto.</echo>

<jar jarfile="\$dist\TesteAnt.jar" basedir="\$build">

<!– Tornando o jar executável–>

```
            <manifest>
                <attribute name="Main-Class" value="TesteAnt"/>
            </manifest>
        </jar>
    </target>
</project>
```