SeleniumTG: UMA FERRAMENTA WEB PARA GERAÇÃO DE SCRIPTS DE TESTE

Trabalho de Conclusão de Curso

Engenharia da Computação

Eduardo Augusto de Oliveira Nazaré

Orientador: Prof. M.Sc. Gustavo Carvalho

Eduardo Augusto de Oliveira Nazaré

SeleniumTG: UMA FERRAMENTA WEB PARA GERAÇÃO DE SCRIPTS DE TESTE

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

Orientador:
Prof. M.Sc. Gustavo Carvalho

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

Recife - PE, Brasil

23 de maio de 2012

ESCOLA POLITÉCNICA
DE PERNAMBUCO

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 29 de 5 de 2012, às 17:00 horas, reuniu-se para deliberar a defesa da monografia de conclusão de curso do discente EDUARDO AUGUSTO DE OLIVEIRA NAZARE, orientado pelo professor Gustavo Henrique Porto de Carvalho, sob título SeleniumTG: uma Ferramenta Web para Geração de Scripts de Teste, a banca composta pelos professores:

Eliane Maria Loiola

Gustavo Henrique Porto de Carvalho

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

☒ Aprovada ☐ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 9,0 (nove)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O discente terá 7 dias para entrega da versão final da monografia a contar da data deste documento.

Eliane

ELIANE MARIA LOIOLA

Gustavo Henrique

GUSTAVO HENRIQUE PORTO DE CARVALHO

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

(Dedicatória)

Dedico este trabalho a todos aqueles
que dedicam sua vida na luta
pelo direitos humanos e coletivos.

Dedico este trabalho a todos aqueles
que compreendem que um mundo melhor
se constrói com uma outra sociedade,
justa e igualitária.

À você que não conheci e conheço tanto, Gregório.

"Gostaria de ser lembrado como o homem
que foi amigo das crianças, dos pobres e excluídos."
Gregório Bezerra

Agradecimentos

Em *O Avesso das Coisas*, o poeta Carlos Drummond de Andrade escreveu: "A amizade é um meio de nos isolarmos da humanidade cultivando algumas pessoas.". E ele está certo, não pelo fato de nos identificarmos mais com umas pessoas do que outras, mas pelo fato de realmente deixarmos de nos identificar com outrem em favor de pequenos círculos.

Dos pequenos círculos de outros, faço o meu grande círculo. Faço o meu primeiro agradecimento àqueles que, anônimos, não costumam ser lembrados no resultado de anos de esforços na academia:

- Aos operários, que construíram a Universidade sob o seu suor e a qual única placa que lhes sobra, injustamente, são póstumas.
- Aos garis, que limpam todo dia nossas vergonhas cotidianas da rua, ignorados e despercebidos na nossa correria do dia-a-dia.
- Aos rodoviários, que me levaram a Universidade todo dia. Que acordam muito antes do amanhecer e dormem muito depois do anoitecer.
- Aos servidores públicos da POLI, que mantêm a máquina tão vital à Universidade pública.

Aos meus amigos de livros e sonhos: o esforço valeu a pena. Quando uma causa é justa, ela não deve ser abandonada. Eu não baixei guarda, apenas juntei forças para o dia seguinte.

Aos meus professores que sei que dedicam todos os seus dias aos seus alunos e à ciência. Faço reconhecer o esforço que fazem para manter e melhorar a qualidade do nosso curso.

A minha mãe, que das adversidades da vida fez sua força. Ela que me lançou escolhas e me deu suporte na minha decisão.

Resumo

Automatização de testes de software é um desejo constante nas empresas de desenvolvimento de software e a plataforma Selenium é um meio para se dar os primeiros passos nessa direção. Contudo, é demasiado custoso manter scripts de teste, devida a mudança de requisitos, tecnologia, etc. Este trabalho apresenta uma ferramenta web de criação de scripts de teste para o Selenium, com seu desenvolvimento gerenciado com metodologia SCRUM. A ferramenta obtém dos fluxos dos casos de uso de um projeto os cenários de teste e, destes, os casos de teste em script HTML para o Selenium.
Palavras-chave: teste, selenium, automação, web, regressão, reteste, ferramenta, caso de uso, scrum.

Abstract

Automated software testing is a constant desire in business software development and platform Selenium is a means to take the first steps in that direction. However, it is too costly to maintain test scripts, due to changing requirements, technology, etc.. This paper presents a web tool for creating test scripts for Selenium, with its managed development with SCRUM methodology. The tool gets the flow of use cases in a project the test scenarios, and of these test cases in HTML script for Selenium.
Keywords: test, selenium, automation, web, regression, retest, use case, scrum.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

Ajax	*Asynchronous Javascript and XML*
DAO	*Data Access Object*
FDD	*Feature Driven Development*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
JDBC	*Java Database Connectivity*
JSF	*Java Server Faces*
J2EE	*Java 2 Platform, Enterprise Edition*
J2SE	*Java 2 Platform, Standard Edition*
MVC	*Model-view-controller*
PO	*Product Owner*
Pojo	*Plain Old Java Objects*
TDD	*Test Driven Development*
WAR	*Web Archive*
Web	*World Wide Web*
XHTML	*Extensible Hypertext Markup Language*
XML	*Extensible Markup Language*

1 Introdução

1.1 Qualificação do Problema

Software tem se tornado não só importante, mas imprescindível para o desenvolvimento da sociedade moderna. Cada vez mais, a sociedade está dependente de sistemas de informação para realizar desde a atividades simples e despercebidas, como controlar um micro-ondas, às complexas e de alta responsabilidade, como controlar um avião.

Tendo contato com variados sistemas, os usuários e clientes de produtos da tecnologia da informação passaram a exigir de forma mais criteriosa a qualidade dos produtos que usam. Deste modo, as empresas desenvolvedoras são obrigadas a cada vez mais serem minuciosas e criteriosas com a qualidade de seus produtos. Falhas, antes corriqueiras em software de décadas atrás, não são mais relevadas pelos usuários, e são também levadas em conta na hora de qualificar e escolher um software. "Sofisticação e complexidade podem causar enormes problemas para quem precisa construir sistemas complexos." (PRESSMAN, 2010).

Dentro da área de qualidade de software e suas subáreas e definições, a disciplina de testes (RATIONAL, 2012) tem crescido a passos menores do que deveria; em parte pela falta de profissionais qualificados e em parte pela ainda fraca cultura de testes nas empresas. De acordo com AQUINO (2012), "as desculpas utilizadas para a não realização dos testes nas empresas são muitas"além de que "a boa notícia é de que há uma expectativa de, nos próximos anos, as empresas mudarão de forma radical a sua postura em relação ao teste de software. Isso porque, a pressão dos usuários por mais qualidade, somada à demanda dos acionistas por produtividade, tende a transformar a questão em uma condição essencial a qualquer projeto de TI.".

Tendo em vista tal espaço para crescimento, as empresas que investem em qualidade – em geral por meio da disciplina de testes – têm tido frutos visíveis e não costumam voltar atrás no investimento, mas sim buscar meios de se desenvolver. AQUINO (2012)

comenta ainda que a economia foi de até 20 vezes para aqueles que acharam defeitos ainda na fase de levantamento de requisitos, de acordo com pesquisa feita com participação de empresas como IBM, GTE e TRW.

PRESSMAN (2010) também observa que "Hoje em dia uma enorme indústria de software tornou-se fator dominante nas economias do mundo industrializado.". Sendo assim, um software de qualidade torna-se fator imprescindível para aqueles que querem posicionar-se a frente do mercado, quer como usuários de soluções, quer como produtores destas soluções. São os dois lados trabalhando mutuamente e interferirindo positivamente no processo de produção de software.

Então, ao encontro desta necessidade de mercado é que temos a esperança nos testes: ampliar garantias em busca da excelência do produto. Sobre isso, acrescenta BASTOS (2007) que as expectativas sobre o teste só podem ser alcançadas quando este é movido sobre um processo próprio, com todas as características de qualquer outro processo, respeitadas suas particularidades.

O teste de software deve ser movido pelos requisitos do cliente e o bom teste é aquele que encontra defeitos (PRESSMAN, 2010). Mas considerando que a complexidade dos programas tem crescido, torna-se inviável buscar – para a grande maioria dos sistemas atuais – cobrir 100% das possibilidades de fluxos e estados. Para isso, devemos utilizar de técnicas próprias de testes que se baseiam em opções representativas dessas possibilidades, buscando diminuir o escopo de teste mas minimizando o prejuízo sobre o resultado (BASTOS, 2007). Ao final, teremos um número reduzido de instâncias de teste (casos de teste) onde então daremos o foco de nossa verificação e análise.

Além de técnicas de redução do escopo, a disciplina de testes utiliza de métodos próprios de execução do trabalho. A vista do usuário, os testes funcionais (BASTOS, 2007) focam no comportamento do sistema e procuram verificar os requisitos. Por si, a natureza destes testes é humana e manual e herda todas as características danosas da operação: custo (recursos humanos, tempo de execução), dispersão do testador, exaustão física, etc. Na Figura 1, temos a representação gráfica do custo de tempo dedicado ao reteste de execução manual ao longo de um projeto.

Contudo, a automação da execução dos testes busca reduzir o esforço e facilitar o trabalho. Com a popularidade da gestão ágil de projeto com (PRESSMAN, 2010), reforça-se a necessidade de retestar casos de teste passados pela justificativa que a unidade de iteração pode promover mudanças substanciais no sistema, invalidando o resultado anterior dos testes. A essa operação repetitiva e retroativa chama-se teste de regressão. (BASTOS,

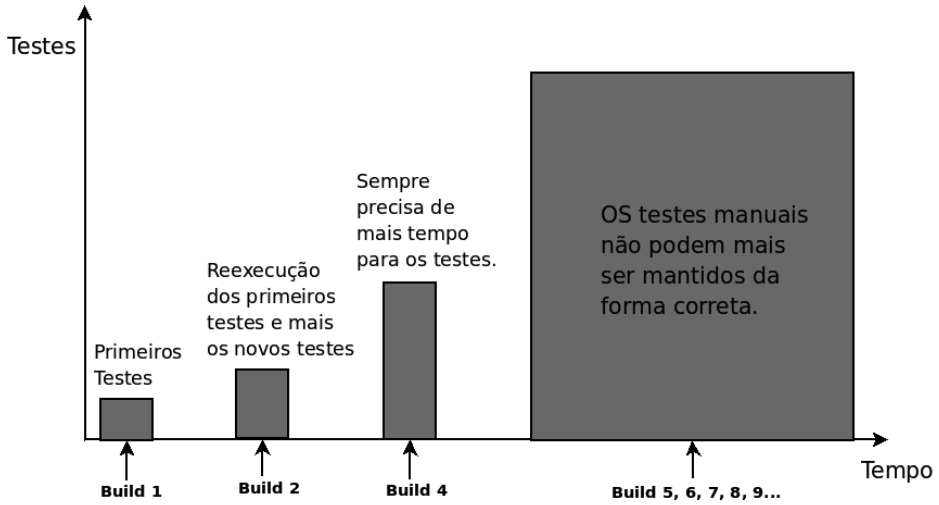

Figura 1: Custo do teste manual *VERSUS* tempo de projeto.
[Fonte: (SARTORELLI, 2007)]

2007)

O custo já visto anteriormente para a execução do teste de regressão por via manual é crescente ao longo do projeto, podendo se tornar um fator de insucesso do mesmo. Já o custo da execução automática destes costuma manter-se dentro de uma média inferior a manual, visto a facilidade e versatilidade da execução dos casos de teste automatizados (scripts de teste). Mostra SARTORELLI (2007), na Figura 2, que o teste automático tem menor custo quando o reteste é constante e necessário.

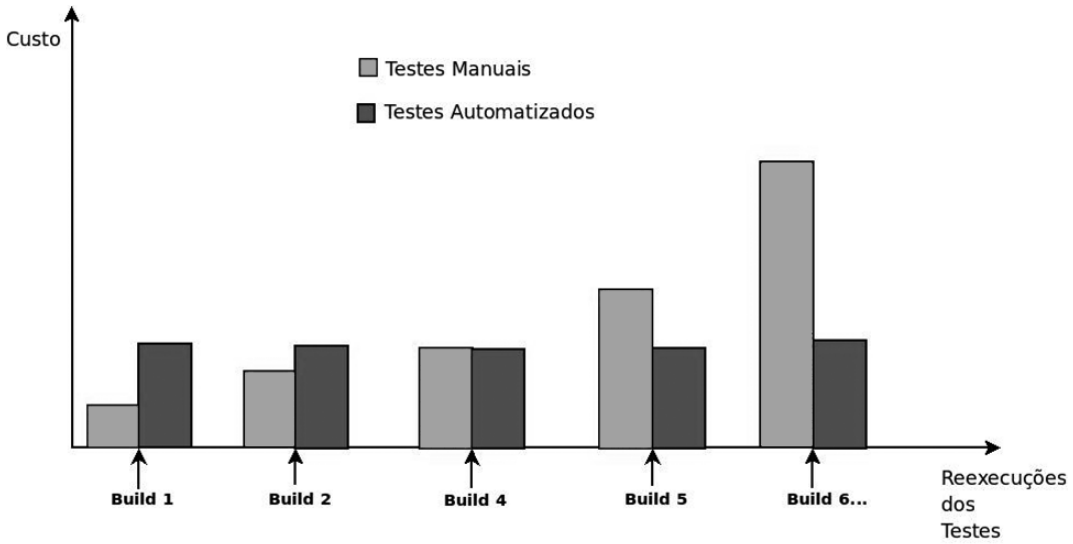

Figura 2: Custo de testes automáticos e manuais ao longo do tempo.
[Fonte: (SARTORELLI, 2007)]

Por outro lado, um problema enfrentado pelas empresas é o alto custo para formação

especializada de uma equipe de automação de testes e ainda pode-se acrescentar a variedade de propostas de ferramentas. Das mais aceitas pelo mercado para a web (*World Wide Web*), o Selenium (SELENIUMHQ, 2012) é uma das propostas mais utilizadas (PATUCI, 2010) em todo o mundo para testes de web funcionais, contando com um conjunto de artifícios poderoso.

Uma limitação a ser observada para a adoção do Selenium permeia duas esferas. A primeira refere-se à criação dos scripts de teste que quase sempre utiliza da gravação de casos de testes manuais pelo plugin com interface gráfica Selenium-IDE (próprio do *framework* Selenium), tornando-se dispendiosa. "Em alguns casos, muito específicos, o *Record and Replay* é muito bem aproveitado, mas ele não deve ser usado em projetos que sofrerão mudanças com o tempo", RIBEIRO (2011). A segunda se refere a manutenção dos scripts de teste gerados, já que torna-se um árduo trabalho editar cada script de teste. Ainda observa RIBEIRO (2011) que o tempo de refatoramento de scripts ruins é cerca de dez vezes o tempo de planejamento.

Este trabalho propõe, portanto, uma ferramenta que estreite a distância entre a equipe de testes e a automação. Tal ferramenta busca facilitar a criação de scripts de testes automáticos por meio de uma interface amigável onde, com o mínimo de conhecimento em testes, é possível construir scripts a partir de descrições em linguagem natural associadas a passos de scripts de testes.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver uma ferramenta web capaz de construir, de maneira facilitada e automática, scripts de teste para a plataforma Selenium a partir de definições textuais dos passos de casos de uso.

1.2.1 Objetivos Específicos

Em particular, a ferramenta pode ter seu objetivo geral desdobrado nos seguintes objetivos específicos:

1. Permitir o mapeamento entre frases e comandos Selenium (i.e., *selenese*, comandos próprios do Selenium);
2. Utilizando as frases mapeadas, permitir descrever o comportamento de um software através de casos de uso que possuem fluxos principais e alternativos;

3. Considerar os cenários de teste identificados a partir dos fluxos principais e alternativos dos casos de uso;

Tal ferramenta deverá ser fácil de ser operada por profissionais de testes, assim, atuando como um facilitador entre a especificação de requisitos e a sua verificação.

1.3 Resultados e Impactos Esperados

O principal resultado deste trabalho é o desenvolvimento de uma ferramenta web que permitirá a geração e manutenção de scripts Selenium, a partir de descrições textuais dos passos de casos de uso.

Espera-se que a ferramenta atue como um facilitador no processo de automação de testes. O mercado deve enxergar que a automação de testes não é uma atividade distante, mas sim viável mesmo no contexto de pequenas empresas.

Com a utilização da ferramenta, espera-se que o custo para manutenção da automação seja reduzido, estimulando, assim, a sua adoção. Essa diminuição de custos se dará pela facilidade de uso da ferramenta, com a possibilidade de gerar e, por conseguinte, manter mais facilmente scripts de teste para Selenium.

1.4 Estrutura da Monografia

A monografia possui os seguintes capítulos:

Capítulo 2: apresenta os conceitos relacionados ao desenvolvimento deste trabalho, bem como trabalhos relacionados.

Capítulo 3: apresenta a metodologia utilizada para o desenvolvimento deste trabalho bem como limitações da pesquisa.

Capítulo 4: apresenta detalhes técnicos do desenvolvimento da ferramenta fruto deste trabalho, como requisitos e particularidades de implementação.

Capítulo 5: conclui este trabalho e apresenta trabalhos futuros.

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

2.1.1 Teste de Software

Teste de software é uma disciplina responsável por prover serviços para outras disciplinas, como requisitos, análise e implantação, enfatizando seu trabalho na avaliação da qualidade do produto de software (RATIONAL, 2012).

Por meio de técnicas próprias de checagem, o teste visa verificar aspectos de requisito e conferir a qualidade do software. Avalia-se o quanto o software está preparado para lidar com situações previstas e não previstas de funcionamento, desde seu ambiente e utilização, até situações extremas. É no teste que situações de exceção deixam evidentes pontos fracos no software e onde se tem desde uma avaliação parcial a uma integral de como o sistema corresponde ao esperado.

Os testes não devem ser deixados para o último momento no desenvolvimento, mas serem utilizados desde o início do ciclo de vida de um software. Atualmente aceitam-se quatro fases de nível de teste: (BASTOS, 2007)

Teste Unitário: feito normalmente pelos desenvolvedores sobre pequenas partes do código.

Teste de Integração: aplicado sobre a junção das pequenas partes do código.

Teste de Sistema: visa conferir o sistema como um todo.

Teste de Aceitação: teste final do sistema para questões de funcionalidade e usabilidade.

Existem duas abordagens de teste de software:

Caixa-preta onde não se tem conhecimento do código-fonte ou da minuciosidade de sua implementação.

Caixa-branca onde se tem acesso livre ao código fonte e detalhes de implementação.

Da primeira abordagem, as técnicas de teste funcional são seus maiores representantes, pois se preocupam em garantir o atendimento aos requisitos, ou seja, que os requisitos estão corretamente codificados (BASTOS, 2007), sem se preocupar com características de implementação. Muito se confunde caixa-preta com teste funcional, mas na verdade os dois têm escopos diversos de análise.

Há de se observar que alguns autores consideram uma terceira abordagem, chamada *Caixa-cinza*. Nesta abordagem, há a combinação das outras duas, tendo o testador acesso ao código-fonte e permitindo, assim, priorizar os testes de acordo com a implementação, podendo priorizar fluxos, condições e situações.

2.1.1.1 Teste Funcional

Por priorizar o software em suas características tanto de requisitos quanto de qualidade, de acordo com PRESSMAN (2010), a técnica de teste funcional busca defeitos em, basicamente, 5 categorias:

1. funções incorretas ou omitidas.
2. erros de interface.
3. erros de estrutura de dados ou de acesso à base de dados externa.
4. erros de comportamento ou desempenho.
5. erros de iniciação e término.

Dividindo-se ainda o teste funcional em outras técnicas mais específicas e objetivas, a técnica de teste funcional desdobra-se, resumidamente, na Tabela 1.

2.1.2 Teste Automatizado

A automação de testes é a mecanização do trabalho repetitivo relacionado aos testes. Queira para planejar, gerir, executar ou analisar, a automatização dos testes busca reduzir

Tabela 1: Técnicas de teste funcional.
[Fonte: (BASTOS, 2007)]

Técnica	Descrição	Exemplo
Requisitos	Determinar se o sistema é executado conforme o especificado	Validação do especificado *versus* implementado. Verificação das políticas e dos regulamentos
Regressão	Verificar se alguma coisa mudou em relação ao que já estava funcionando corretamente	Mudança de segmentos funcionais do sistema. Mudança manual de procedimentos corretos
Erro no manuseio (usabilidade)	Prevenir ou detectar erros para depois consertá-los	Falha introduzida no teste. Erro reincidente
Suporte manual	Preparar os dados para processamento e usar os dados fornecidos pelo sistema	Informação de *input*. Tomada de ações baseadas nas informações contidas nos relatórios
Integração	Assegurar que a interconexão entre as aplicações funciona corretamente	Cenário de teste de integração
Controle	Analisar que a interconexão entre as aplicações funciona corretamente	Cenário de teste de integração
Paralelo	Comparar resultados do aplicativo atual com a versão anterior	Comparação dos resultados obtidos com as versões nova e antiga para garantir a equivalência

esforços e dar respostas mais rápidas a atividade de testes, reduzindo custos. Em muitos casos é imprescindível para se chegar a resultados rápidos e claros. Contudo, se tratadas sem a devida análise, considerações e respeito aos limites, podem corresponder ao inverso do esperado, trazendo mais prejuízos do que benefícios.

Da automação de testes, a automação pela utilização de ferramentas de apoio é uma das principais alternativas. Para CAETANO (2007), ferramentas servem para dar suporte a todas as atividades relacionadas ao ciclo de vida de desenvolvimento de software: da concepção à implantação. Fortalece e pondera BASTOS (2007), que veículo para executar um processo de teste, a ferramenta é um recurso para o testador e, sozinha, é insuficiente para conduzir todo o teste. (...) o martelo é apenas uma ferramenta, que, sem a técnica para seu uso, nem sempre será eficaz.

É na execução dos testes que a automação se torna mais evidente. Para os software em camadas, existem uma infinidade de ferramentas para cada uma das camadas. Mas é na camada de apresentação que os testes funcionais permeiam, em testes como usabilidade, interface, funcionalidade, etc.

2.1.2.1 Selenium

Selenium é uma suíte de ferramentas para automação de testes web (SELENIUMHQ, 2012). Com ela é possivel desde gravar ações do usuário e verificar informações da tela,

como também programar suítes de teste com integração em várias linguagens.

"O Selenium é uma ferramenta *Open Source* usada para a criação de testes de regressão automatizados para aplicações web. O Selenium foi escrito utilizando Java Script e DHTML. Em função disso, os testes rodam diretamente a partir do navegador." (CAETANO, 2007).

Por meio de uma linguagem própria de comandos, incluindo ações de usuário e verificações do sistema, é que o Selenium executa seus testes. Esta "linguagem"própria é chamada *selenese*. Na verdade, trata-se de um conjunto de comandos que incluem *actions* (ações do usuário, como clicar em um botão), *accessors* (funções de verificações do tipo 'get' e 'is', como pegar o valor de um campo) e *assertions* (funções de afirmação que podem falhar o teste, como conferir se o título de uma página está correto). Os dois últimos podem inclusive ser criados dinamicamente, de acordo com a necessidade do testador quando utilizada alguma linguagem de programação suportada pelo Selenium.

O Selenium conta, atualmente, com um conjunto que inclue quatro ferramentas principais:

Selenium 2: é o projeto futuro do Selenium, que se encontra ainda em fase de desenvolvimento. Integra características antigas da suíte com uma maior flexibilidade por meio da fusão com o WebDriver[1].

Selenium IDE: é um plugin para o navegador Firefox. Registra e reproduz interações com o navegador (como *click* em link e preenchimento de formulários) e facilita a alteração e adaptação dos scripts.

Selenium 1: resulta da especialização do Selenium Remote Control (Selenium RC). Foi o principal projeto do Selenium, até a fusão com o WebDriver. É um servidor escrito em Java, necessário para execução de scripts de teste do Selenium RC, suportando as linguagens de programação Java, Javascript, Ruby, PHP, Python, Perl e C#.

Selenium Grid: diferente do Selenium 1, permite que sejam executados múltiplos testes em paralelo, em diferentes máquinas e sistemas operacionais, utilizando processamento em paralelo.

O *core* do Selenium funciona com base na *selenese*. Quando chamada para executar scripts HTML, o Selenium faz *deploy* do arquivo HTML, buscando, entre outras coisas,

[1]WebDriver: projeto da Google.

pelo endereço raiz de execução do script e os passos a serem executados, indiferente se são ações do usuário ou comandos de verificação e validação.

Um exemplo de um script de teste pode ser visto na Tabela 2. Este exemplo tem como objetivo abrir a tela com informações sobre leitos do Portal de Dados Abertos do Governo de Pernambuco.

Tabela 2: Exemplo de script de teste Selenium em HTML.

Busca por "leitos"		
open	http://www.dadosabertos.pe.gov.br	
type	id=tbBuscaInput	leitos
clickAndWait	id=btBusca	

Na primeira linha da Tabela 2, temos o nome do caso de teste que serve como referencial para manutenção e visualização do arquivo. Nas linhas seguintes temos as seguintes ações, em sequência:

1. Abertura da página do Portal de Dados Abertos.
2. Escrita do texto "leitos"no campo de busca.
3. Clique no botão de submissão da busca e aguardo da resposta.

Ao final da execução do script de teste, temos o navegador na tela esperada. Esse script é simples e pode ser extendido, fazendo validação de alguns campos, valores, etc.

O script de exemplo pode ser executado no Selenium de várias formas. Pelo Selenium 1, há opção de usar o *Test Runner*, mostrado na Figura 3, que é uma interface web para rodar suítes de teste HTML.

Deste modo, o Selenium é um dos automatizadores de teste funcional web mais utilizados em todo o mundo (PATUCI, 2010), sendo foco de pesquisa deste trabalho.

2.2 Trabalhos Relacionados

Na direção deste trabalho, outras ferramentas buscaram automatizar a geração e manutenção de testes, com distintas metodologias. Contudo, poucas mostraram-se comparáveis a este trabalho. Esta seção apresenta algumas destas ferramentas disponíveis.

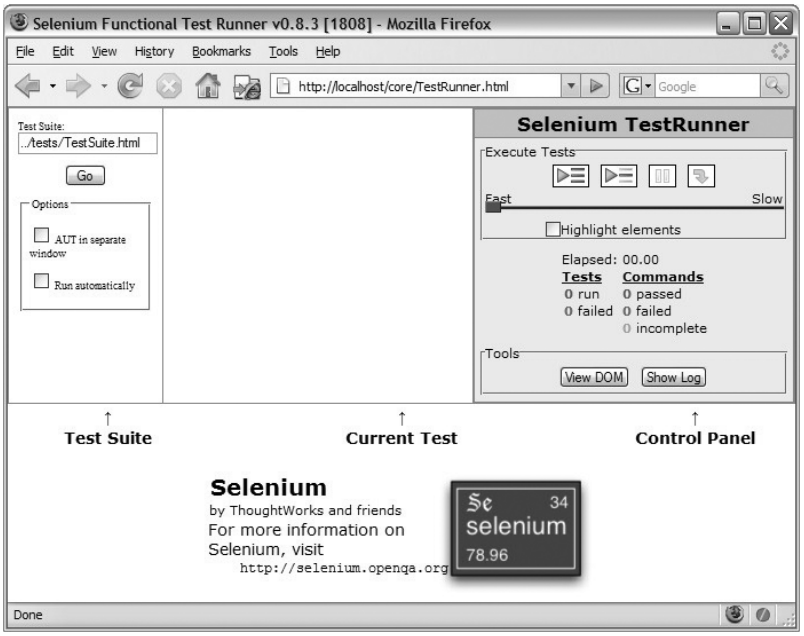

Figura 3: Tela do *Test Runner* sendo executado no navegador Firefox.
[Fonte: (SELENIUMHQ, 2012)]

2.2.1 UCT - Use Case Tester

Em SILVA (2011) tem-se uma avaliação da UCT - Use Case Tester (CARNIELLO ADRIANA; JINO, 2005), uma ferramenta de cobertura de testes preparada para suportar mudanças. A ferramenta resume-se em três funcionalidades: determinação dos requisitos, simulação do teste e análise dos resultados. Baseada por critérios dados pelo testador, são criados os casos de teste e a ferramenta percorre os grafos de possibilidades de testes e avalia inconscistências.

A UCT não gera casos de teste e muito menos scripts de teste, funciona como um avaliador de requisitos pelos fluxos encontrados para o caso de uso, podendo simular a execução dentro das possibilidades de caminhos dentro dos fluxos. Logo, não supre as necessidades impostas nesse trabalho por não gerar casos de teste e muito menos os scripts de teste para o Selenium.

2.2.2 TaRGeT

A TaRGeT (*Test and Requirements Generation Tool*) tem como objetivo a criação de cenários de teste com base em um documento de casos de uso, formatado a um modelo XML. Com esta ferramenta é possível criar casos de teste rapidamente e com filtros variados, como cobertura e requisitos, por meio de uma interface para *desktop* (SIQUEIRA,

2010).

Ela é *open-source* e conta com módulos muito úteis que foram agregados com o tempo, de acordo com a necessidade das empresas e pesquisas que com ela trabalharam e acabaram contribuindo com a ferramenta. Seus principais módulos são:

Editor de Caso de Uso: funciona como um facilitador de edição e também adição dos casos de uso para um projeto nela incorporado.

Controlled Natural Language: é um módulo capaz de identificar ambiguidades descritas nos casos de uso, utilizando uma linguagem natural controlada.

On The Fly Generation: é o principal módulo, ou *core* da ferramenta, onde os casos de teste são gerados de acordo com filtros diversos, como caso de uso, propósito, cobertura, etc.

A Figura 4 mostra a TaRGeT em operação em um projeto fictício com a tela de geração de casos de teste e seus diversos filtros.

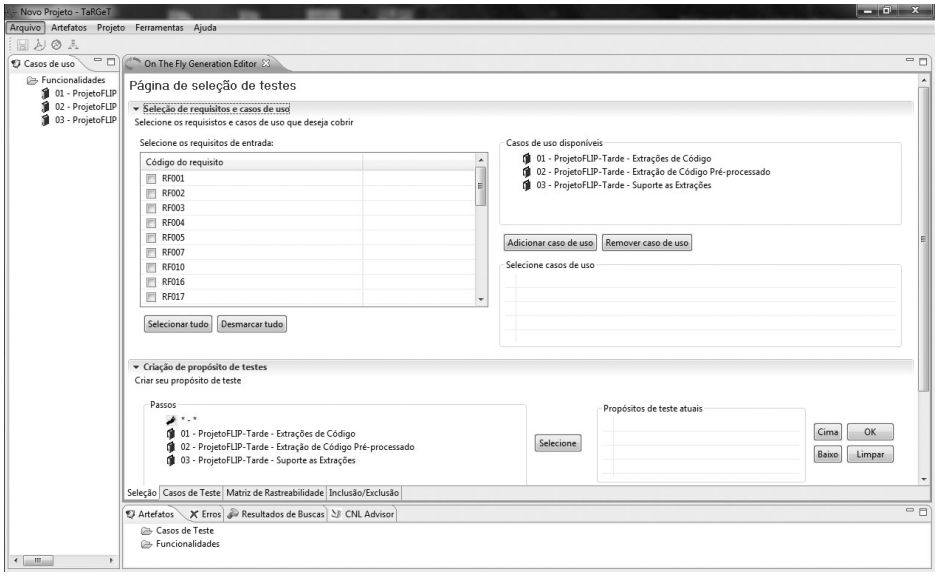

Figura 4: Tela da TaRGeT.

Ela é uma ferramenta robusta para geração e manutenção de casos de teste. Das ferramentas apresentadas, a que mais se assemelha com o proposto por este projeto. Contudo, toda a automação da ferramenta é para testes manuais, não tendo integração para criação de scripts de teste.

3 Método de Pesquisa

As próximas seções descrevem a metodologia aplicada no desenvolvimento deste trabalho.

3.1 Qualificação do Método de Pesquisa

O desenvolvimento da ferramenta proposta por este trabalho será guiado por um processo de gerenciamento próximo ao SCRUM, já que tem como proposta ser "um processo ágil que permite manter o foco na entrega do maior valor de negócio, no menor tempo possível." (BROD, 2012). Essa metodologia visa gerir projetos de maneira ágil, com retorno rápido ao cliente e focado no que o cliente tem como mais prioritário para o projeto (por exemplo, alguma funcionalidade).

Não deve-se confundir *metodologia de gerência ágil* com *metodologia de desenvolvimento ágil*. O SCRUM é uma metodologia de gerência ágil de projeto, podendo ser aplicado, em tese, a qualquer projeto (de software ou não) onde uma metodologia ágil torne-se interessante. Já as metodologias de desenvolvimento ágil como TDD (*Test Driven Development*) e FDD (*Feature Driven Development*) (PRESSMAN, 2010), visam trazer agilidade ao desenvolvimento de um software, sendo íntimamente ligado a disciplina de implementação (RATIONAL, 2012) e com foco em técnicas ágeis para implementação. Como as duas têm como característica a agilidade, são fortes candidatas a serem utilizadas em conjunto, ou seja, pode-se usar SCRUM com FDD.

Pela metodologia aplicada, consideramos a adaptação dos seguintes envolvidos (BROD, 2012):

Product Owner é a pessoa envolvida com o resultado do projeto, normalmente chamada simplesmente de PO. Em geral, é o próprio cliente. No caso, assumiremos o orientador deste trabalho com esta função.

Team ou Time, é toda a equipe responsável por encaminhar o projeto e fazer suas entregas acontecerem. Nesse caso, a equipe é formada por uma pessoa, autor deste trabalho.

Scrum Master é um membro da equipe responsável por assegurar que a prática do SCRUM seja seguida e não ocorra desvios e coloque o resultado em risco. Função também assumida pelo autor deste trabalho.

Atrelado ao desenvolvimento da ferramenta tem-se a produção dos seguintes artefatos (incluindo os próprios artefatos do SCRUM):

Backlog: é uma lista de requerimentos de funcionalidades da ferramenta descrita pelo PO. Ela contém em alto os requisitos do projeto, dividido em histórias *User Stories*.

User Stories: são histórias com definições textuais dos requisitos, dentro de um modelo simples e facilitado, e funcionalidades do sistema. Sintetizam, em poucas palavras, um conjunto de expectativas sobre a ferramenta. São importantes para nortear a construção da ferramenta.

Diagrama de Casos de Uso: provê mais detalhes do que as *User Stories* em relação ao comportamento esperado da ferramenta. Já leva em consideração questões analíticas dos passos de execução.

Diagrama de Classes: exibe as associações entre as classes, tais como generalizações, agregações, entre outras.

Diagrama Entidade-Relacional: descreve o software quanto a persistência dos dados no banco de dados, com suas propriedades e relações entre entidades.

Cronograma: cronograma de todo o projeto, desde a elicitação até a entrega.

A Figura 5 exibe um gráfico representativo do ciclo da metodologia SCRUM. Nela, temos o *backlog* como entrada para as *sprints*, com reuniões diárias e projetos (ou partes funcionais do projeto) com entrega de poucos meses. Estas *sprints* são as unidades básicas de desenvolvimento pela metodologia do SCRUM, usualmente de uma a quatro semanas. Terminada uma *sprint*, há uma reunião de avaliação do resultado e planejamento da próxima.

O SCRUM permite que, de forma ágil, o trabalho possa ser dividido, estimado e executado, com acompanhamento do PO. Com adaptações feitas, o autor deste trabalho

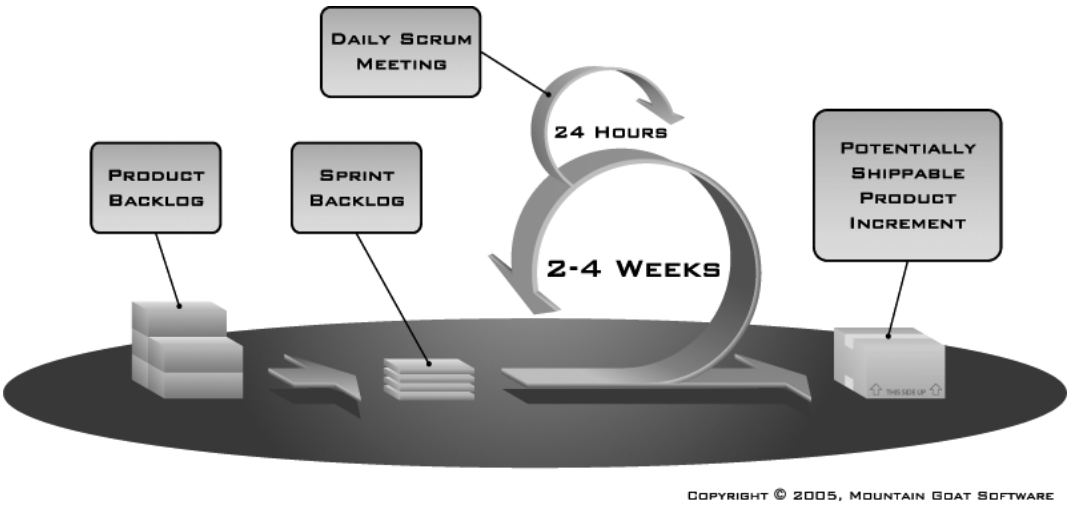

Figura 5: Representação do ciclo da metodologia SCRUM.
[Fonte: (BROD, 2012)]

ficará responsável por sua execução ("time"de uma pessoa) e reavaliação de prioridades a cada semana, fim da *sprint*.

Desse modo, funcionalidades serão elicitadas a partir da necessidade proposta pelo PO. Além disso, o progresso será acompanhado semanalmente através de reuniões presenciais.

3.2 Etapas do Método de Pesquisa

Em conjunto com o método de pesquisa, este trabalho teve como etapas:

Estudo de Conceitos Relacionados: pesquisa bibliográfica e estudo sobre conceitos de engenharia de software, com foco engenharia de testes. Pesquisa e estudo do *framework* Selenium.

Análise de Trabalhos Relacionados: pesquisa sobre ferramentas existentes similares a proposta deste trabalho.

Projeto e Implementação: projeto e implementação de uma ferramenta web com as características descritas neste trabalho, necessárias a solução do problema de pesquisa levantado.

Avaliação Preliminar: avaliação da ferramenta por meio de testes práticos de comportamento e resposta.

3.3 Limitações da Pesquisa

Este trabalho teve limitações intrínsecas e adquiridas, com a finalidade de diminuir o escopo apresentado para tornar viável a conclusão do trabalho dentro do prazo disponível. Segue uma lista das limitações da ferramenta proposta:

1. A ferramenta necessita de uma configuração própria, proposta para desenvolvimento Java Web por LUCKOW D. H.; MELO (2010), adaptado para as novas versões dos instrumentos utilizados. Estes são citados no Capítulo 4.

2. Não foi integrada a ferramenta deste trabalho um algoritmo de geração de cenários e casos de teste, tornando-se escopo externo ao trabalho proposto.

3. Os casos de uso que são utilizados pela ferramenta cumprem a necessidade e adaptações próprias, não sendo foco da ferramenta ser um mantenedor de casos de uso. Para os casos de uso utilizados, assumiremos as propriedades de nome, descrição, pós-condição, pré-condição, um fluxo principal e vários fluxos alternativos. Para cada fluxo terei passos, que incluem ação do usuário e do sistema.

4. Não foi executado um experimento para comprovação do resultado esperado na ferramenta.

4 Resultados

Visto no Capítulo 3 o método utilizado neste trabalho e com base no referencial apresentado no Capítulo 2, este capítulo discorre sobre a ferramenta resultado. Inicialmente é motrado os requisitos e modelagem da ferramenta. Posteriormente é mostrado uma visão funcional da solução.

4.1 Requisitos

Os requisitos levantados para a ferramenta, pela metodologia SCRUM, incluíram critérios de validação das funcionalidades e comportamento, incluindo protótipos.

A ferramenta proposta foi delineada por funcionalidades, baseadas nos requisitos funcionais dado pelo PO e descritas em um *backlog*.

Não tornou-se escopo deste trabalho a confecção de algoritmo capaz de gerar os cenários de teste a partir dos casos de uso, citando-o como trabalho futuro no Capítulo 5.

4.2 Backlog

Aqui temos os ítens do *backlog* de requisitos e funcionalidades assim como foram levantados com o PO.

#01 - Como um **Usuário**
eu posso/gostaria/devo **acrescentar frases como passos**
para **serem utilizadas nos passos dos casos de uso**
Validação:

- Os passos aceitam constantes e variáveis.
- Cada passo associa-se a uma lista de 'n' comandos Selenium.

- As frases podem ser do tipo Ação do Usuário ou Verificação do Sistema.

#02 - Como um **Usuário**
eu posso/gostaria/devo **apagar frases**
para **para não serem mais utilizadas**
Validação:

- Permitir a remoção de frases, mesmo que inclusas em algum caso de uso.

#03 - Como um **Usuário**
eu posso/gostaria/devo **acrescentar projetos na ferramenta**
para **gerenciar todos os testes da ferramenta**
Validação:

- Receber nome, descrição e URL base.

#04 - Como um **Usuário**
eu posso/gostaria/devo **remover projetos na ferramenta**
para **remover toda sua informação**
Validação:

- Solicitar confirmação de exclusão.

#05 - Como um **Usuário**
eu posso/gostaria/devo **adicionar casos de uso**
para **servir de base para a geração dos cenários de teste**
Validação:

- Associar o caso de uso a um projeto.
- Receber descrição, pré-condição, pós-condição e associar uma lista de 'n' passos como fluxo principal.

#06 - Como um **Usuário**
eu posso/gostaria/devo **adicionar fluxos alternativos a um caso de uso**
para **servir de base para a geração dos cenários de teste, junto com o fluxo principal**
Validação:

- Receber: descrição, pré-condição, pós-condição e uma lista de 'n' passos.
- Tomar como ponto de partida um passo do fluxo principal e sua condição de verificação de sistema para entrada. Tomar como retorno um passo do fluxo principal.

#07 - Como um **Usuário**
eu posso/gostaria/devo **criar casos de testes**
para **execução**
Validação:

- Os casos de testes serão criados a partir da associação de um cenário de teste com uma tupla de dados para as varáveis daquele cenário.

#08 - Como um **Usuário**
eu posso/gostaria/devo **criar suíte de testes**
para **execução**
Validação:

- O usuário escolherá quais os casos de teste farão parte da suíte de testes.
- A execução dos testes se dará por suíte e não por casos de testes diretamente.

#09 - Como um **Usuário**
eu posso/gostaria/devo **salvar uma suíte de testes**
para **execução dos casos de teste da suíte**
Validação:

- A suíte deve vir em arquivo único compactado.

#10 - Como um **Usuário**
eu posso/gostaria/devo **acessar a página inicial da ferramenta**
para **informações iniciais e introdutórias da ferramenta**
Validação:

- Obedecer protótipo.

4.3 Protótipos de Tela

Em reuniões com o PO, requisitos foram levantados e resumidos no *backlog*. Este, foi alterado e concluído com a prototipação das telas, as quais nesta seção será mostrada.

Na expectativa de facilitar trabalhos futuros com base nesta ferramenta e de acordo com exigência do PO, a prototipação ocorreu na língua inglesa para as telas, apesar da implementação ter ocorrido em língua portuguesa, mas com fácil adaptação para qualquer língua, como será mostrado posteriormente.

As telas prototipadas podem ser visualizadas no Apêndice A.

4.4 Casos de Uso

Com base nos protótipos de tela, o diagrama de caso de uso pode ser representado na Figura 6.

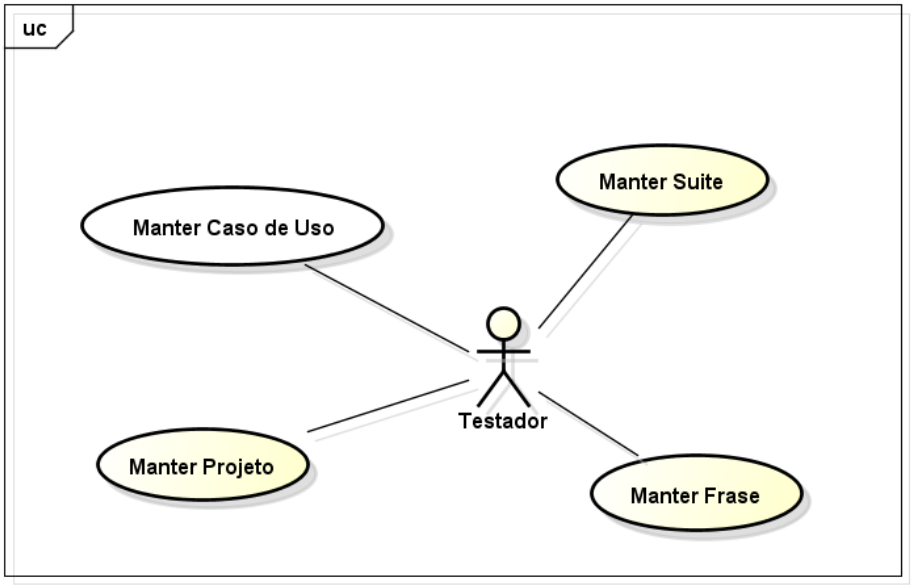

Figura 6: Diagrama de casos de uso da ferramenta.

4.4.1 Descrição de Casos de Uso

Os casos de uso descritos na Figura 6 têm as seguintes descrições:

UC01: Manter Projeto

Objetivo: Gerenciar os projetos da ferramenta.

Pré-condição: Estar na tela de gerência de projetos.

Pós-condição: Os projetos foram gerenciados.

Fluxo Principal

Usuário

1. O usuário procura um projeto na lista de projetos.

Sistema

1. O sistema exibe o projeto selecionado.

Fluxo Alternativo 1 – Adicionar um Projeto

Entrada/Saída: 1/FIM

Pré-condição: O projeto não foi encontrado.

Usuário

1.1. O usuário preenche pelo menos os campos obrigatórios Nome e URL do novo projeto.

1.2. O usuário clica em Salvar.

Sistema

1.1. O sistema exibe os campos preenchidos na tela.

1.2. O sistema cria um novo projeto e passa a exibi-lo na lista de projetos.

Fluxo Alternativo 2 – Remover um Projeto

Entrada/Saída: 1/FIM

Pré-condição: O projeto foi encontrado.

Usuário

2. O usuário clica em Excluir.

3. O usuário confirma.

Sistema

2. O sistema exibe uma mensagem de confirmação.

3. O sistema remove o projeto do sistema e não mais passa a exibi-lo na lista de projetos. Todas as associações ao projeto são também excluídas.

UC02: Manter Caso de Uso

Objetivo: Gerenciar os casos de uso de um projeto

Pré-condição: Estar na tela de gerência de casos de uso.

Pós-condição: Os casos de uso foram gerenciados.

Fluxo Principal

Usuário

1. O usuário seleciona um projeto para serem listados os casos de uso.

2. O usuário procura um caso de uso na lista de casos de uso e seleciona Editar.

3. O usuário altera os dados do caso de uso.

4. O usuário clica em Salvar.

Sistema

1. O projeto é selecionado e o sistema passa a listar os casos de uso daquele projeto.

2. O sistema apresenta as informações do caso de uso selecionado.

3. O sistema apresenta o caso de uso com as alterações.

4. O sistema salva as alterações do caso de uso e limpa os campos da tela.

Fluxo Alternativo 1 – Adicionar um caso de uso

Entrada/Saída: 1/FIM

Pré-condição: O caso de uso não foi encontrado.

Usuário

1.1. O usuário preenche os campos obrigatórios.

1.2. O usuário clica em Adicionar passo ao Fluxo Principal.

1.3. O usuário seleciona um passo de ação e um de verificação.

1.4. O usuário clica em Adicionar.

1.5. O usuário clica em Salvar.

Sistema

1.1. O sistema exibe as informações do caso de uso.

1.2. O sistema exibe uma popup de seleção das frases do passo.

1.3. O passo de ação e verificação são destacados na lista.

1.4. O sistema adiciona o novo passo ao fluxo e fecha a tela de adicionar passos.

1.5. O sistema salva o novo caso de uso e deixa de apresentar na tela as informações entradas. O novo caso de uso passa a ser listado.

Fluxo Alternativo 2 – Remover um Caso de Uso

Entrada/Saída: Passo 1/FIM

Pré-condição: Não estar editando um caso de uso.

Usuário

2. O usuário procura um caso de uso na lista de casos de uso e seleciona Excluir.

3. O usuário confirma.

Sistema

2. O sistema exibe uma mensagem de confirmação.

3. O sistema remove o caso de uso do sistema e não mais passa a exibi-lo na lista de casos de uso do projeto. Todas as associações daquele caso de uso são também excluídas.

Fluxo Alternativo 3 – Adicionar um Fluxo Alternativo

Entrada/Saída: Passo 3/4; Passo 1.4/1.5;

Pré-condição: Estar editando ou adicionando um caso de uso.

Usuário

4. O usuário clica em adicionar fluxo alternativo.
5. O usuário clica em Editar do fluxo alternativo.
6. O usuário preenche os campos Descrição, De, Quando e Para.
7. O usuário clica em Adicionar passo ao Fluxo Principal.
8. O usuário seleciona um passo de ação e um de verificação.
9. O usuário clica em Adicionar.
10. O usuário clica em Salvar.

Sistema

4. O sistema adiciona um fluxo alternativo à lista de fluxos alternativos.
5. O sistema exibe uma popup para edição do fluxo alternativo.
6. O sistema exibe os campos preenchidos.
7. O sistema exibe uma popup de seleção das frases do passo.
8. O passo de ação e verificação são destacados na lista.
9. O sistema adiciona o novo passo ao fluxo e fecha a tela de adicionar passos.
10. O sistema atualiza o fluxo alternativo na lista de fluxos alternativos.

UC03: Gerar Suíte

Objetivo: Gerar casos de teste e suítes de teste.

Pré-condição: Ter cenários criados a partir dos casos de uso.

Pós-condição: Uma suíte de teste foi gerada.

Fluxo Principal

Usuário

1. O usuário seleciona o cenário para gerar caso de teste e clica em Novo Caso de Teste.
2. O usuário clica em Editar para cada dado necessário e entra as informações necessárias.
3. O usuário confirma a edição de cada dado necessário.
4. O usuário clica em Salvar.
5. Na lista de casos de teste o usuário seleciona quais casos de teste deseja para uma nova suíte de testes.
6. O usuário clica em Criar Nova Suíte.
7. O usuário entra o Nome da suíte e clica em Salvar.
8. O usuário clica em Baixar a Suíte.

Sistema

1. O sistema apresenta um popup para entrada dos dados necessários a criação do caso de teste para aquele cenário.
2. O sistema exibe as informações entradas.
3. Os campos de dados são preenchidos de acordo com o entrado.
4. O sistema fecha o popup de criação do Caso de Teste e passa a listar o caso de teste adicionado.
5. Os casos de teste são selecionados na lista.
6. O sistema exibe uma popup para entrada do nome da suíte a ser criada.
7. O sistema gera a suíte de testes com base nos casos de teste e passa a listá-la na lista de suítes.
8. O sistema envia a suíte compactada.

Fluxo Alternativo 1 – Remover um Caso de Teste

Entrada/Saída: Passo 4/5

Pré-condição: Estar na tela de suítes.

Usuário

4.1. O usuário clica em Remover para um caso de teste listado.

4.2. O usuário confirma a remoção do caso de teste.

Sistema

4.1. O sistema exibe uma tela de confirmação.

4.2. O sistema exclui o caso de teste e ele não passa mais a ser listado.

UC04: Manter Frases

Objetivo: Gerenciar as frases da ferramenta

Pré-condição: Estar na tela de gerência de frases.

Pós-condição: As frases foram gerenciadas.

Fluxo Principal

Usuário

1. O usuário seleciona a aba do tipo de frase.
2. O usuário procura uma frase cadastrada na lista de frases e a seleciona.

Sistema

1. O sistema lista as frases do tipo selecionado.
2. O sistema destaca a frase selecionada e apresenta suas informações.

Fluxo Alternativo 1 – Adicionar uma frase

Entrada/Saída: Passo 2/FIM

Pré-condição: Estar na tela de frases.

Usuário

2.1. O usuário entra com a descrição da frase com seus parâmetros

2.2. O usuário clica em adicionar um comando Selenium.

2.3. O usuário seleciona um comando.

4.5 Arquitetura e Padrões de Projeto

De acordo com LUCKOW D. H.; MELO (2010), "um sistema de fácil manutenção é um sistema em que as várias responsabilidades estão separadas e implementadas em locais bem-definidos.". Deste modo, responsabilidades como de tela ou acesso ao banco de dados serão tratados em camadas diversas e especializadas. Isso garante um melhor controle e modularidade do software, aspectos importantes de qualidade.

Para separação da camada de acesso dos dados foi aplicado o padrão de projeto DAO (*Data Access Object*), especializando algumas classes para somente realizarem as operações no banco de dados. As camadas para esta finalidade seguirão o padrão de interface "*Pojo*DAO"[1]. A Figura 7 mostra um gráfico representativo do padrão.

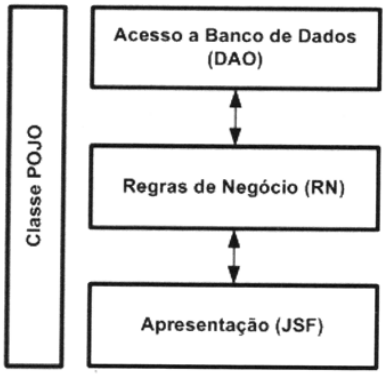

Figura 7: Representação do padrão DAO.
[Fonte: (LUCKOW D. H.; MELO, 2010)]

As classes POJO, mostradas na Figura 7 são as classes elementares do programa. Para acesso à camada de dados, as classes DAO nos dão interfaces que podem ser implementadas de diferentes maneiras para diferentes tecnologias. As regras de negócio são tratadas em classes específicas, anteriores (quanto ao fluxo de entrada de dados) às classes DAO. Já a apresentação fica a cargo das páginas JSF, onde temos a camada de apresentação.

[1]POJO: *Plain and Old Java Object.* Classes bases independentes de interface ou framework.

Já o controle de regras de negócio não foi tratado na camada de acesso aos dados, pois isso não cabe a esta camada. As solicitações que chegam a camada de acesso aos dados já devem chegar tratadas e validadas na camada de regra de negócio, que é onde estão tratadas as regras, condições e onde são lançadas exceções necessárias para tratamento. Essas classes seguirão o padrão "*Pojo*RN".

O padrão *Singleton* foi utilizado nas classes *Bean*[2], reduzindo o número de instâncias criadas e consultas ao banco de dados por meio do controle de instâncias do tipo lista de dados (métodos *get*).

Depois de levantados os requisitos iniciais para o projeto, foi aceita a arquitetura MVC (*Model-view-controller*) proposta em (LUCKOW D. H.; MELO, 2010) para produtos Java Web.

O padrão MVC é tanto um padrão de arquitetura quanto de projeto e é dos mais utilizados para sistemas web. Ele orienta a utilização de três camadas para o projeto:

Model: camada de acesso a dados e regras de negócio.

View: camada de interface visual do sistema. É nela que o resultado da camada *Model* é mostrado.

Controller: camada que faz a conecção das outras duas camadas.

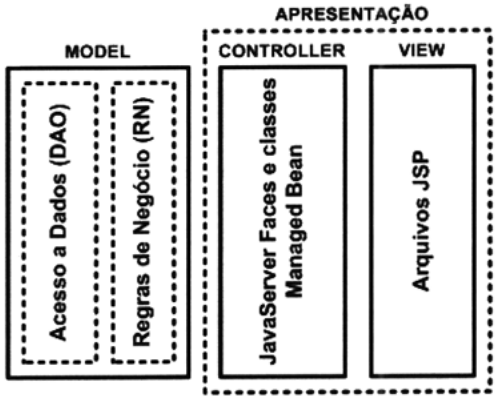

Figura 8: Representação do padão MVC.
[Fonte: (LUCKOW D. H.; MELO, 2010)]

Por este padrão, fazem parte da camada *Model* as classes de acesso aos dados (DAO), de regras de negócio (RN) e as mais básicas (POJO). Já as outras duas camadas, *View*

[2]Bean: classes que suportam o funcionamento das páginas JSF e permeiam a camada de visualização e de dados.

e *Controller*, estão ligadas a apresentação, incluindo as classes *Bean* para controle e JSP para saída do código HTML.

O que mais caracteriza o padrão MVC no desenvolvimento da ferramenta é tanto a separação de responsabilidades quanto a definição de como as camadas deverão interagir, facilitando e simplificando o entendimento da arquitetura.

4.6 Diagrama Conceitual

A Figura 9 exibe o diagragrama de classes, resultado da modelagem conceitual necessária para a solução do problema.

No diagrama, o projeto é a classe que determina a existência das demais entidades. Para cada projeto, tenho vários casos de uso e um caso de uso faz parte de um e somente um projeto.

O caso de uso tem como atributos um fluxo principal e vários alternativos. Para um fluxo, principal ou alternativo, tenho uma sequência de passos. Cada passo, por sua vez tem duas frases relacionadas: uma diz quanto a ação do usuário e outra diz quanto a resposta do sistema, de acordo com o modelo definido.

Cada frase associa-se a um comando do Selenium por meio de uma classe associativa, necessária por existir atributos intrínsecos a relação das duas entidades. Há também uma classe de identificação desta relação, importante para perfeita normalização das entidades no banco de dados.

Aparentemente há uma repetição de atributos de nome *parâmetro*, mas esses atributos são necessários como meio de mapear o que um parâmetro de um comando se refere quando é utilizado numa frase e posteriormente num passo, facilitando a utilização da ferramenta pelo usuário. Sem tal mapeamento por meio de replicações até a utilização do passo pelo usuário, não haveria referencial para uma usuário criar uma frase ou passo.

A geração dos cenários, dá a um caso de uso vários cenários com base no fluxo principal e seu *embaralhamento* com os fluxos alternativos. Um caso de teste se distingue de um cenário, por ser uma instância deste, mas contendo dados específicos.

Por fim, a classe de suíte é uma entidade sem ligação direta com as demais, pois ela só tem sentido para a camada de visualização, pois sua persistência é física, em arquivos HTML.

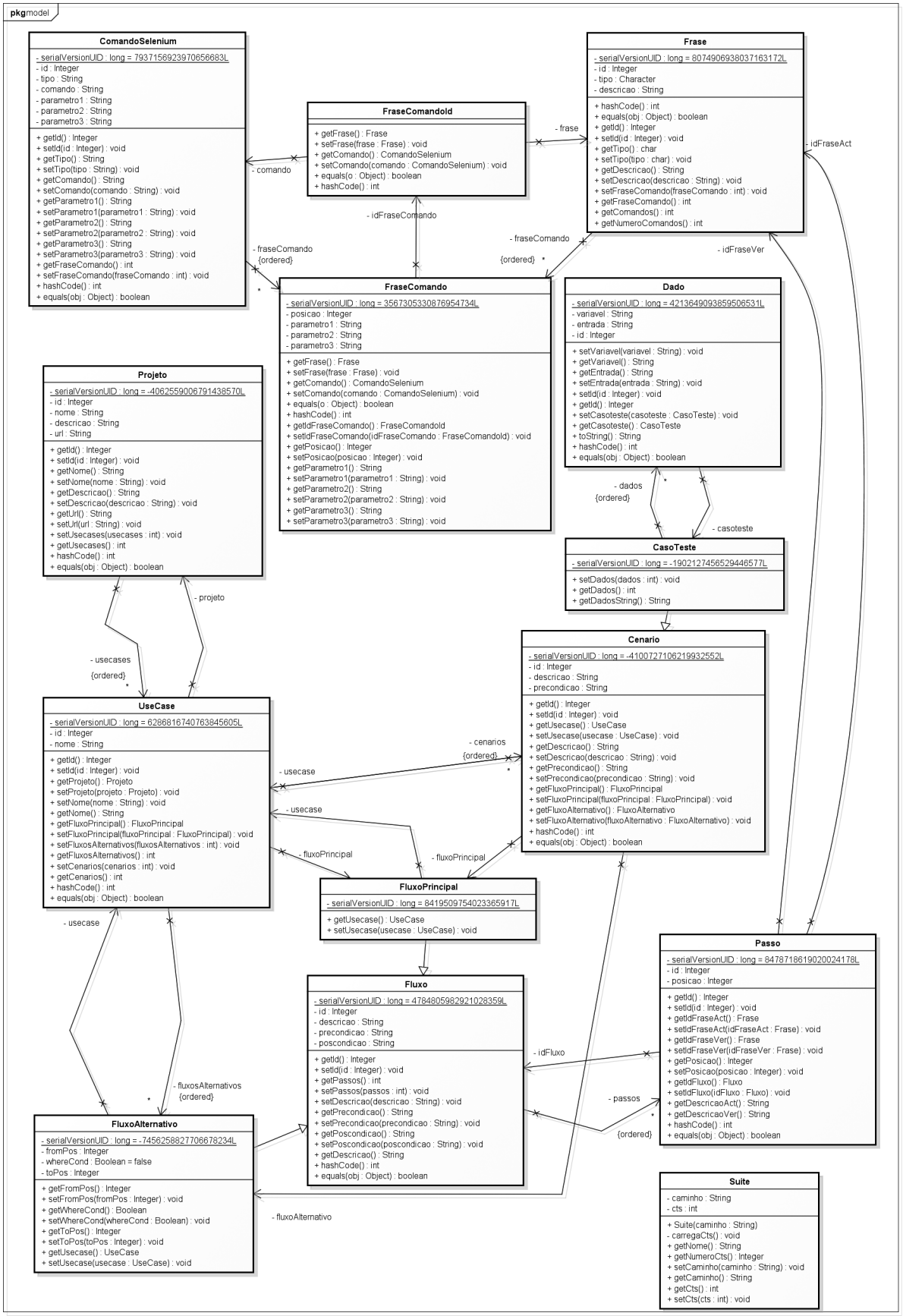

Figura 9: Diagrama de Classes.

4.7 Diagrama Entidade-Relacional

A Figura 10 exibe o Diagrama Entidade-Relacional (E-R) para persistência dos dados. O diagrama mostra as relações existentes e necessárias, destacando-se que as suítes de testes não terão persistência em banco de dados, mas física com arquivos HTML prontos para serem executados, por isso não estão inclusas no diagrama.

O diagrama E-R exibe algo muito próximo do diagrama de classes, devido a facilidade de mapeamento entre os dois diagramas com a utilização da tecnologia descrita em LUCKOW D. H.; MELO (2010) e mencionada ainda neste capítulo.

Foi procurada a normalização e simplicidade de representação das entidades no banco de dados, resultando em somente uma entidade para cenário/caso de teste e três entidades para representar fluxos principais e alternativos. Este último, tomado assim devido as particularidades do fluxo alternativo em relação ao principal, como passo de entrada, condição de entrada e passo para retorno ao fluxo principal.

4.8 Tecnologia

Esta seção explica como as tecnologias Java Web, JSF, Primefaces, Tomcat e Hibernate foram usadas na ferramenta.

4.8.1 Java Web

Java Web é como é conhecida o uso da tecnologia Java (**??**) para aplicações web. Ela utiliza a plataforma J2EE (*Java Enterprise Edition*), que é uma extensão do *core* da linguagem, o J2SE (*Java Standar Edition*), muito conhecido em aplicações desktop. Herda, assim como toda a sua linha de plataformas, todas as características da linguagem.

O J2EE permite a linguagem trabalhar com aplicações corporativas e distribuídas, baseadas em componentes. É capaz de suportar um grande número de usuários, sendo utilizado em sites de grandes empresas, como as Lojas Americanas, a companhia aérea TAM, o Banco do Brasil e o portal 4Shared.

Também, a J2EE difere muito pouco do J2SE. Suas particularidades estão nas entidades próprias e pacotes utilizados, como por exemplo as classes implementadas para a camada de controle (*Bean*).

O J2EE necessita de um servidor de aplicações (*application server*), que ficará encar-

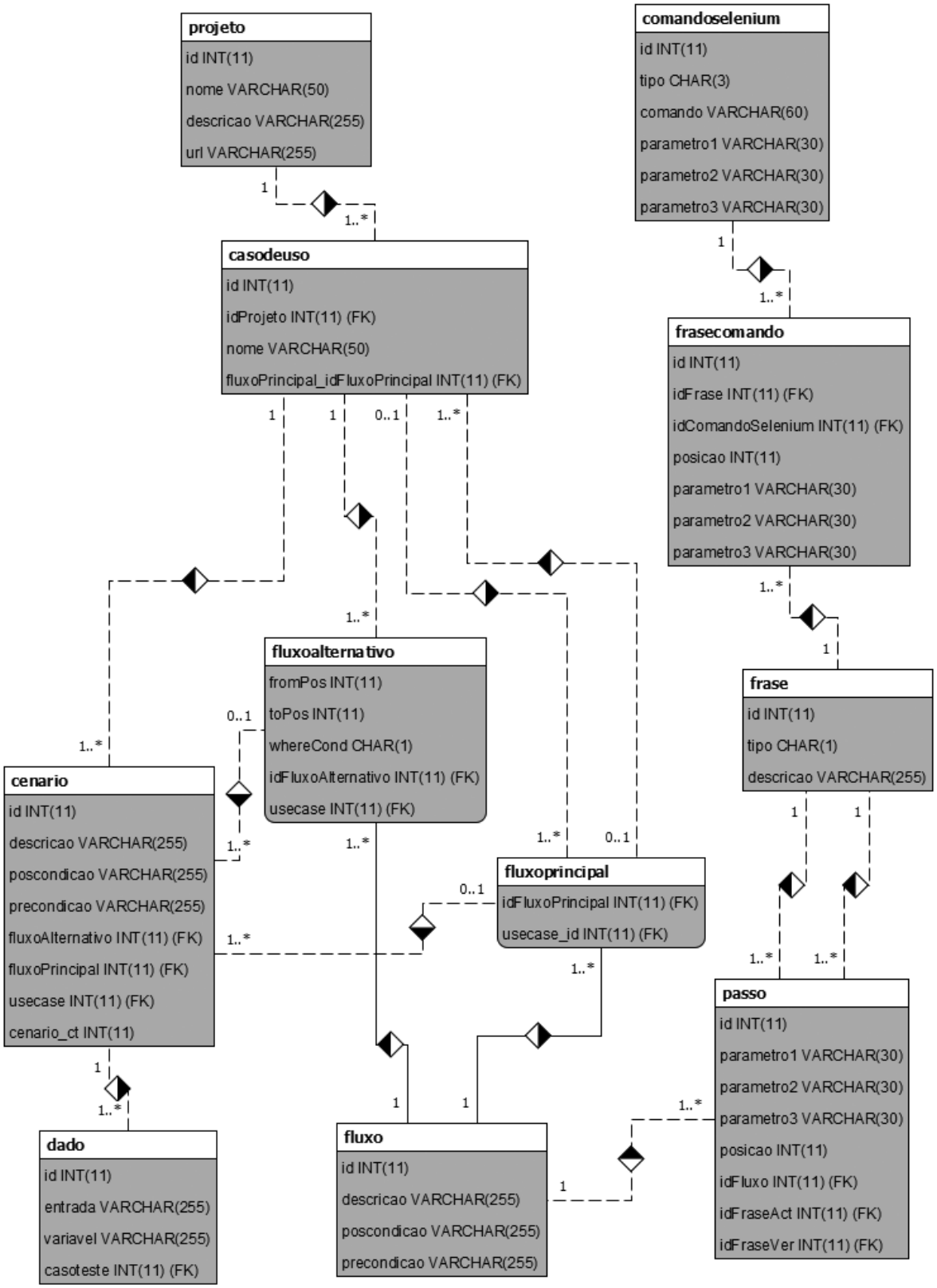

Figura 10: Diagrama Entidade-Relacional.

regado de desempacotar o aplicativo WAR (*Web Archive*) da ferramenta e instanciar a aplicação e suas entidades. Este arquivo WAR nada mais é do que um arquivo ZIP (compactado) com toda a estrutura de diretórios e arquivos da ferramenta, incluindo arquivos JAVA, HTML, imagens, etc. Isto tornou a ferramenta portátil e somente dependente de qualquer programa servidor de aplicações.

4.8.2 JavaServer Faces

O JSF (*JavaServer Faces*) é um *framework* de desenvolvimento Java Web, definido pelo *Java Community Process*[3] (JCP). Grandes empresas, como Apache, Fujitsu, IBM e Siemens participam da especificação do JSF. Não é uma plataforma completa, mas permite extensões importantes, como o PrimeFaces. (LUCKOW D. H.; MELO, 2010)

O JSF suportou as páginas XHTML (*eXtensible Hypertext Markup Language*) da ferramenta. É nessas páginas onde ficam o código HTML com algumas formatações XML, como código para listar todos os projetos cadastrados.

O PrimeFaces[4] é uma extensão do JSF, disponibilizando mais de 90 componentes de fácil utilização. A maioria dos componentes utilizam da tecnologia Ajax (*Asynchronous Javascript and XML*), permitindo uma melhor interação do usuário com a aplicação, tornando um diferencial importante para muitas aplicações. Desde as tabelas, às *popups*, mensagens de aviso, etc., todas as tarefas que normalmente são aplicadas sobre código HTML simples, foram transferidas para o Primefaces, tornando possível, em especial, a atualização de listas de projetos, casos de uso, passos, suítes, etc., na tela sem ter que recarregar toda a página.

O uso trouxe uma dinâmica particular a ferramenta, com janelas dentro da própria tela da página, mensagens de alerta no topo, *design* arrojado para tabelas, seleção de linhas de tabela.

4.8.3 Tomcat

O Tomcat (Apache Tomcat) é um contêiner Java de *servlets* e servidor web, desenvolvido pela *Apache Software Fundation*. Suporta tecnologias *Java Servlet* e JSP. Apesar de suportar um ambiente de produção, pode ser integrado ainda com o Apache HTTP e IIS

[3] Java Community Process: entidade organizada pelo mercado para definições da linguagem Java, além das definidas pela dona, Oracle.

[4] Primefaces: pode ser obtido na url http://www.primefaces.org

da Microsoft para poder aumentar sua capacidade. (LUCKOW D. H.; MELO, 2010)

É o servidor de aplicações utilizado no desenvolvimento da ferramenta, bastando cerca de 10 segundos de *start* para que a ferramenta esteja ativa e preparada para ser utilizada. O Tomcat ativa a escuta de portas para requisições HTTP (*Hypertext Transfer Protocol*), deixando a ferramenta pronta para tratar as requisições e fazer a máquina JAVA funcionar sobre a ferramenta.

4.8.4 Hibernate

A persistência de informação da ferramenta ficou a cargo do banco de dados MySql, que supriu totalmente a necessidade e mostrou-se bastante flexível pelo modelo lógico a qual é baseado.

O Java, para conectar-se a um banco de dados, utiliza-se de um *driver* responsável de fazer a intermediação deste com o Java. É exatamente o JDBC (*Java Database Connectivity*) que define como se comportará o Java para operações sobre um banco de dados.

Como forma de buscar simplificar e diminuir a distância do modelo de persistência com o modelo de classes, o Hibernate foi utilizado para mapear as classes com as entidades do banco de dados. Este Hibernate inclue um conjunto de soluções para tratamento de persistência, onde por meio de arquivos de configuração é possível facilmente montar toda a estrutura do modelo de banco, de acordo com o implementado pelas classes.

A Figura 11 exibe um diagrama com as principais tecnologias usadas na ferramenta, de acordo como foram aplicadas.

No diagrama, vemos que o usuário interage com a ferramenta por meio do resultado do JSF/Primefaces, que é um código XHTML interpretado pelo navegador. A ferramenta construída sobre J2EE é suportada pelo Tomcat, que mantém o aplicativo ativo. A persistência dos dados fica a cargo da abstração de alto nível do Hibernate.

4.9 A Ferramenta SeleniumTG

A Figura 12 ilustra o fluxo de utilização da ferramenta. Ela resume o fluxo básico para utilização da ferramenta.

Para tornar a ferramenta mais amigável, ela conta com uma página inicial de apresen-

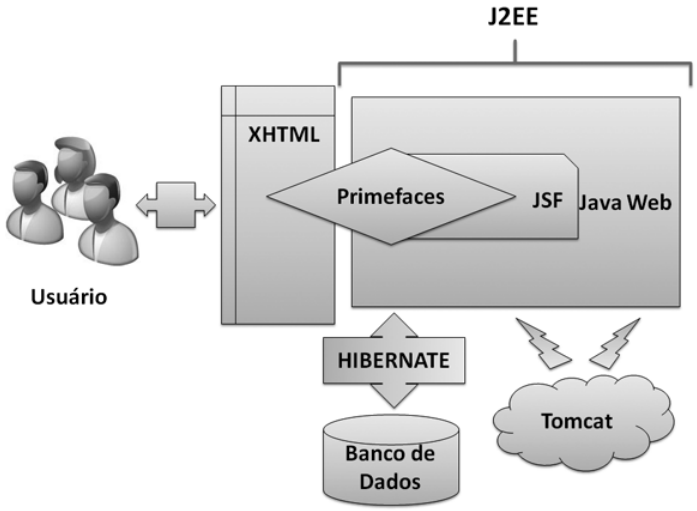

Figura 11: Diagrama de interação de tecnologias.

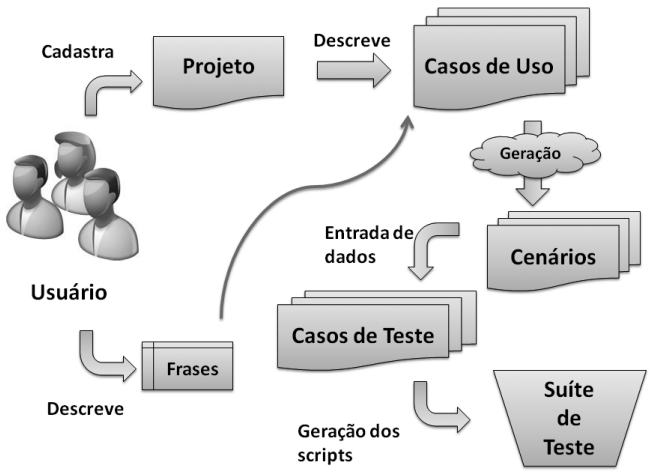

Figura 12: Diagrama de fluxo da ferramenta.

tação, mostrada na Figura 13. Essa página não faz parte da funcionalidade da ferramenta, mas contém informações introdutórias dos propósitos da mesma.

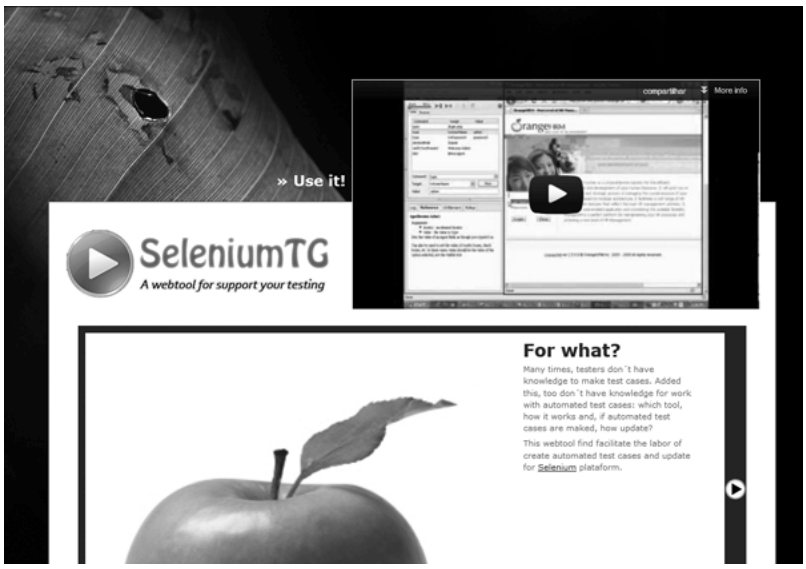

Figura 13: Tela inicial da ferramenta.

4.9.1 Utilização

A primeira tela de utilização exibe a tela de configuração dos projetos cadastrados. A Figura 14 exibe esta tela. No topo da tela, há um menu para acesso às principais funcionalidades descritas pelos casos de uso, bem como uma caixa para escolha de qual projeto será editado. Os campos de nome e URL são obrigatórios. Já o campo de descrição é opcional.

É nesta tela onde o usuário pode gerenciar os projetos existentes, incluindo novos projetos ou excluindo algum cadastro.

Cadastrado um projeto, o usuário passa para a tela de gerência de casos de uso, como mostrado na Figura 15, onde os casos de uso podem ser adicionados, excluídos ou editados.

Por meio desta tela o usuário poderar associar fluxos e passos aos fluxos do caso de uso, entre fluxo principal e fluxo alternativo. A ferramenta aceita que cada caso de uso tenha um fluxo principal e vários fluxos alternativos. O campo nome é obrigatório e os demais são opcionais. A Figura 16 mostra a *popup* para escolha das frases relacionadas a um passo do fluxo. De um lado temos a escolha do passo de ação do usuário e de outro alguma verificação que queiramos fazer após a ação. Abaixo, há um botão para adição de

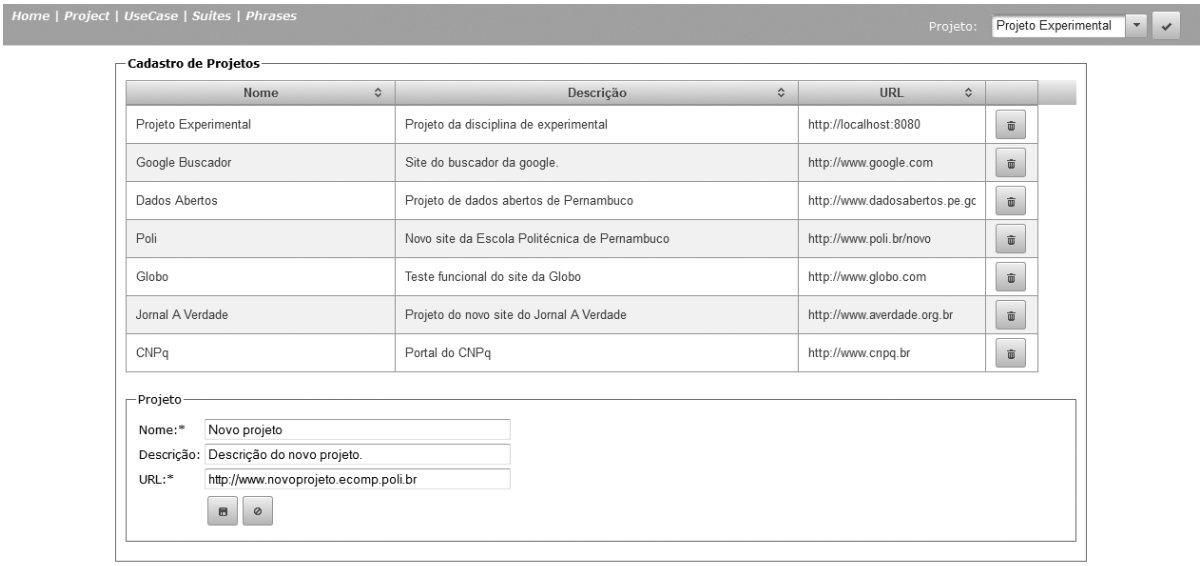

Figura 14: Tela de gerência de projetos.

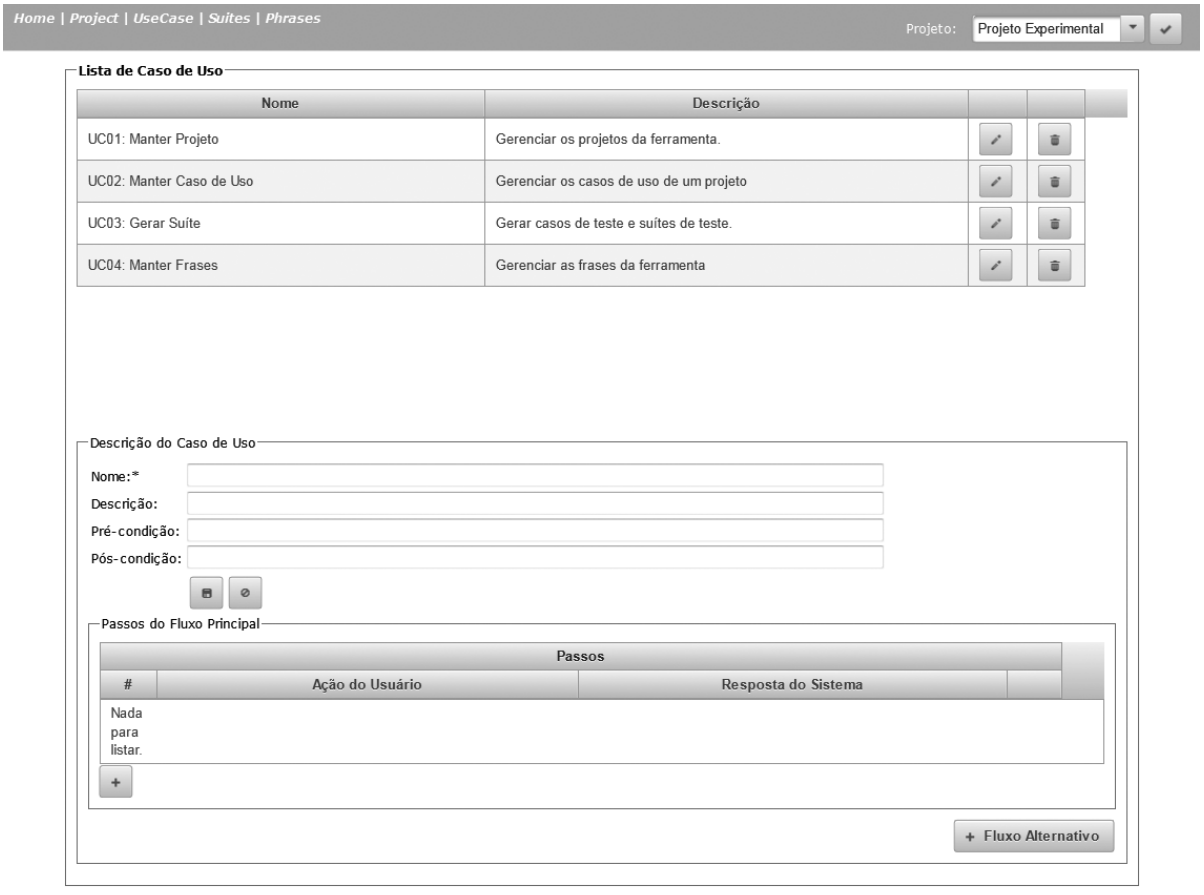

Figura 15: Tela de gerência de casos de uso.

fluxos alternativos que são listados abaixo do botão. Para um fluxo alternativo listado, há a funcionalidade de editá-lo.

Figura 16: Popup de adição de passo a um fluxo.

O sistema considera como variáveis, passíveis de serem informados dados para cada script de teste, as palavras iniciadas e terminadas em "$". Já para as palavras iniciadas e terminadas com "#", o sistema considera como constantes, passíveis de serem informados dados para todo o cenário a que for criado, ou seja, todos os casos de teste assumiram o mesmo valor para a constante.

Para um fluxo alternativo, o usuário pode optar pela condição a que o fluxo alternativo está submetido sob o fluxo principal. Ou seja, um fluxo alternativo pode ser definido como um ramo do fluxo principal. A Figura 17 deixa claro como foi tratado o fluxo alternativo pela ferramenta. Na tela, temos os campos como opcionais. O campo "De"determina a qual passo do fluxo principal o fluxo alternativo está ramificando e o campo "Quando"sob que condição. Já o campo "Para"diz para que passo do fluxo principal o fluxo alternativo irá retornar.

Figura 17: Popup de edição de um fluxo alternativo.

Adicionados os casos de uso, a ferramenta, quando integrada a um algoritmo de

geração de cenários, gerará cenários com base nos fluxos possíveis para cada caso de uso. O número de cenários por caso de uso é dependente da complexidade ciclomática para a árvore de caminhos possíveis. Esta está citada como trabalho futuro no Capítulo 5. Neste trabalho os cenários foram elaborados e adicionados a ferramenta para possibilitar a demonstração da geração dos respectivos scripts de teste.

Após os cenários criados, o usuário tem na tela de manutenção das suítes a possibilidade de criar casos de teste e suites com os casos de teste criados. Para cada cenário podem ser criados infinitos casos de teste, bastando apenas informar os dados do caso de teste que se deseja criar. Estes dados necessários incluem desde informações para localização dos elementos quanto valores para campos. A Figura 18 exibe a tela de manutenção de suítes ao solicitar a criação de um caso de teste com base em um cenário.

Para a ferramenta, um caso de teste se destingue de um cenário pela existência dos dados. Um caso de teste se distingue de outro tanto pelo cenário de teste quanto pelos dados entrados.

O Selenium permite a localização de elementos da página (campos, texto, botões, etc.) pelos atributos *id* ou *name*, pelo caminho *XPath*, por *link*, pelo caminho *DOM* ou ainda pelo caminho de *CSS Selector* (SELENIUMHQ, 2012).

Para as suítes criadas, a ferramenta permite que seja feito o *download* a mesma compactada em arquivo tipo ZIP, vindo os arquivos tipo HTML para execução pelo Selenium. Os arquivos compactados estão em uma pasta com o nome da suíte, contendo um arquivo com referência a cada arquivo de caso de teste da suíte.

Por último, de acordo com a necessidade do usuário, a ferramenta permite gerir as frases cadastradas. Essa ação é a única da ferramenta que necessita claramente de envolvimento de um usuário com conhecimento técnico em *selenese*. Cada frase é uma associação com vários comandos da *selenese*, abstraindo do usuário que for utilizar o restante das funcionalidades da ferramenta o conhecimento de Selenium, como seus comandos. Na Figura 19 é exibida a tela de gerência das frases.

Nesta tela, o usuário tem uma lista dos comandos da *selenese* reconhecidos, para facilitar a familiarização. Para captar esses comandos, foi implementado um *parser* sobre a *url* da última atualização[5] da referência de comandos do Selenium. Este pode retornar em qualquer *console* de desenvolvimento as inserções necessárias para popular o banco de dados de comandos Selenium. Deste modo a ferramenta torna-se facilmente extensível a

[5]URL: http://release.seleniumhq.org/selenium-core/1.0.1/reference.html

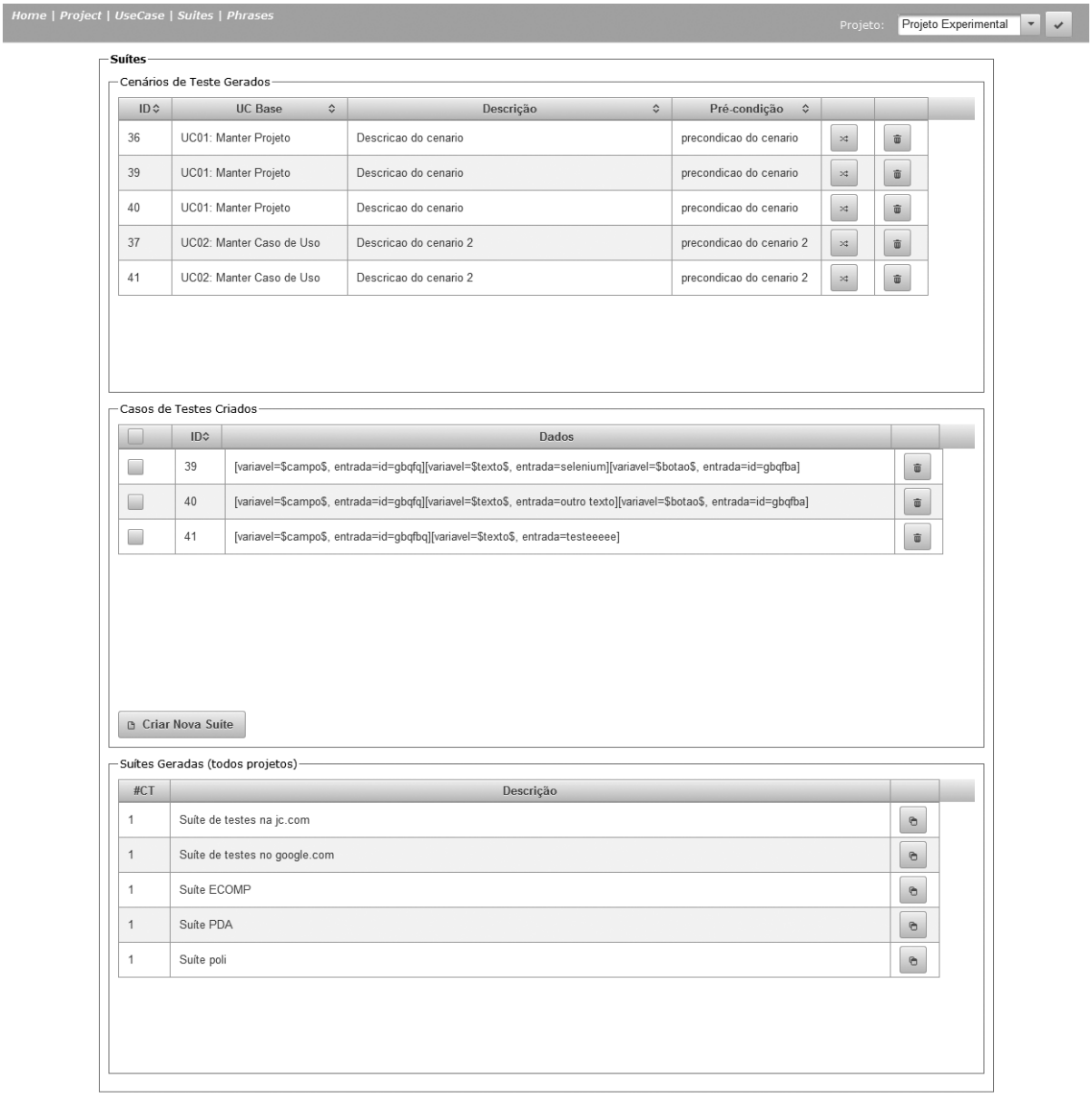

Figura 18: Tela de geração de suítes.

versões futuras do Selenium.

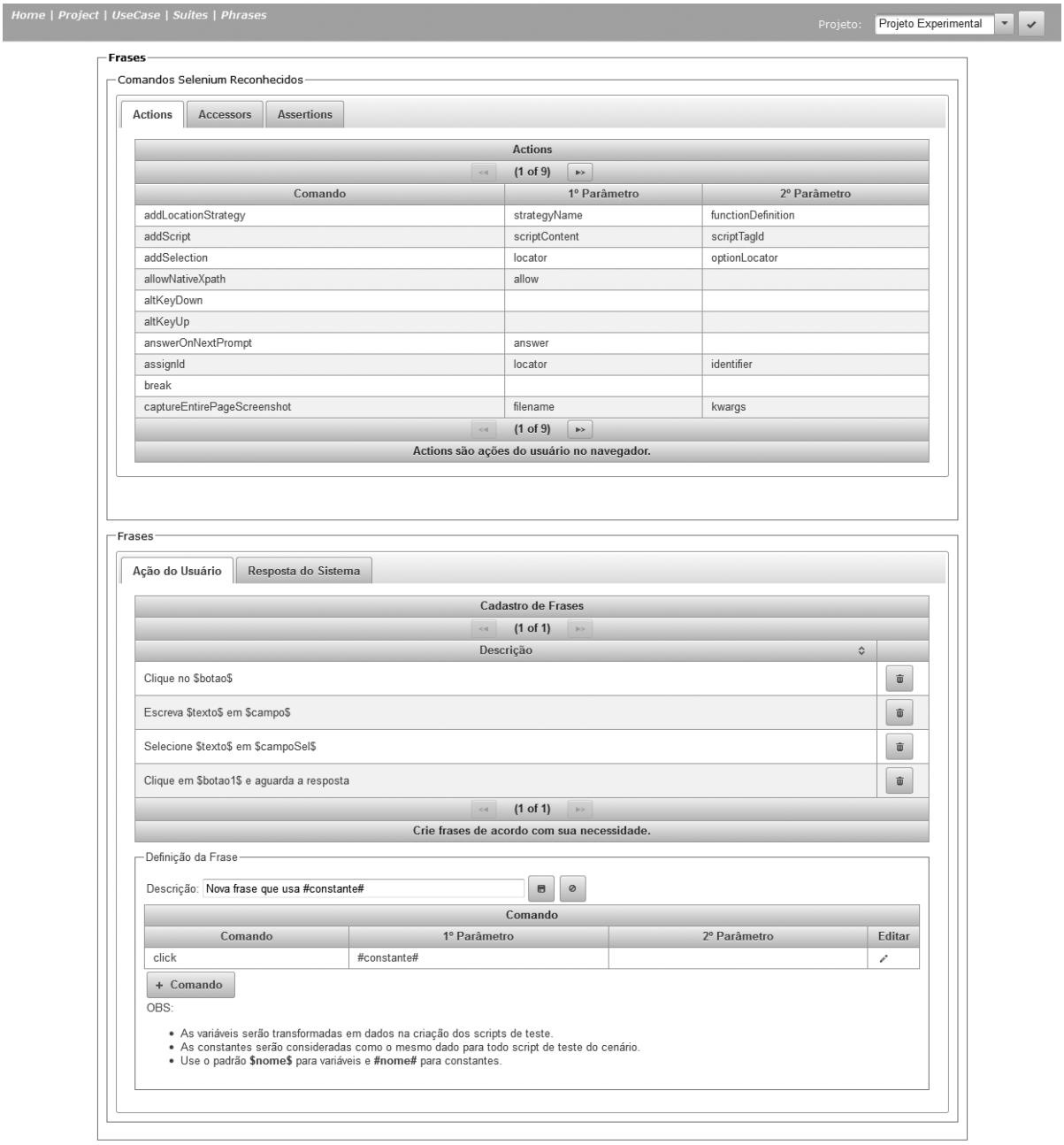

Figura 19: Tela de gestão das frases.

5 Considerações Finais

5.1 Conclusões

Testes de software se caracterizam pelo trabalho por vezes repetitivo e investigativo do testador para procurar encontrar defeitos antes mesmo do cliente ou do usuário. Com mudanças e adaptações ao longo do desenvolvimento de um projeto, os testadores têm que arrumar meios de garantir que o que já foi testado continua com os mesmos resultados. Neste sentido, a automação torna-se interessante, pois consegue trazer características "mecânicas"(automáticas) em uma atividade que cresce junto com o projeto.

A automação de testes de software visa reduzir o esforço de execução dos testes – que antes se concentraria na repetição manual, acumulativa e demorada – para então trazer meios automáticos e rápidos, facilitando a execução de testes de regressão. Automatizar, contudo, não é uma tarefa fácil. Requer conhecimento específico, dedicação, interesse e, para alguns, sensibilidade para identificar que testes podem ser automatizados.

Ainda, para se manter scripts de teste em larga escala e submetidos a mudanças de variáveis como tecnologia e ambiente, automatizar testes pode ser muito mais caro do que a realização manual.

Baseado nesta visão, este trabalho teve como objetivo desenvolver uma ferramenta que reduzisse o tempo para manutenção de scripts de teste para plataforma Selenium, facilitando a geração e manutenção. Deste modo, espera-se que da disciplina de testes mais tempo para a análise dos resultados da execução, tornando a manutenção menos dispendiosa e cansativa.

O resultado mostrou-se promissor, podendo tornar-se um *framework* para geração de scripts de teste, bastando integrar algoritmos de geração de scripts de teste para a tecnologia que se desejar.

Das dificuldades enfrentadas no trabalho, mostra-se a falta de padronização de termos técnicos de teste. Para alguns estudiosos, o conceito de cenário de teste por este trabalho

utilizado é tratado como caso de teste. Já o conceito de caso de teste aqui assumido é uma instância do caso de teste. Contudo, após pesquisa, foi assumido o conceito descrito em (BASTOS, 2007).

Também, não foi encontrado um framework para desenvolvimento utilizando-se o conceito de "caso de uso/cenário/caso de teste". Este conceito teve que ser implementado sem referências práticas, levando muito tempo para ser concluído. Contudo, o resultado aplicado na ferramenta pode ser refinado e utilizado como um *framework* para objetivos similares.

5.2 Trabalhos Futuros

Pela pesquisa levantada, alguns trabalhos futuros são propostos:

1. Pesquisar e integrar a ferramenta com um algoritmo de geração de casos de teste. Com apenas uma tela e um algoritmo eficaz e extensível, é possível integrar a ferramenta.

2. Melhorar a ferramenta quanto aos critérios de validação de entrada do usuário e usabilidade. Muitos campos têm regras de negócio terminaram por não ser tratados na ferramenta.

3. Permitir um controle de acesso e perfil para os usuários. É importantíssimo para a qualidade da ferramenta que ela tenha um controle de acesso e permissões.

4. Executar experimento a fim de avaliar as conformidade das expectativas da ferramenta.

5. Permitir que a ferramenta execute os scripts de teste automaticamente, por meio de integração com o Selenium Webdriver. É possível ter na própria ferramenta uma tela de execução e completar o ciclo de testes automáticos, utilizando o Primefaces.

6. Prover a ferramenta de condições para geração de scripts em linguagens das suportadas, como Java e Python.

Referências

AQUINO, A. *O gargalo da qualidade*. 2012. Web. Acessado em: 11/04/2012. Disponível em: <http://www.tiinside.com.br/11/04/2012/o-gargalo-da-qualidade/ti/272569/news.aspx>.

BASTOS, A. e. a. *Base de conhecimento em teste de software*. 2. ed. São Paulo, Brasil: Ed. Martins, 2007. 263 p.

BROD, C. *Uma introdução ao SCRUM*. 2012. Mountain Goat Software, LLC. Acessado em 11/04/2012. Disponível em: <http://www.mountaingoatsoftware.com/scrum-a-presentation>.

CAETANO, C. *Automação e Gerenciamento de Testes - Aumentando a Produtividade com as Principais Soluções Open Source e Gratuitas*. Julho 2007. 1a Edição v2.0 (atualizada em Outubro/2007).

CARNIELLO ADRIANA; JINO, M. C. M. L. Structural testing with use cases. *Journal of Computer Science & Technology*, v. 5, n. 2, Agosto 2005.

LUCKOW D. H.; MELO, A. A. d. *Programação Java para a Web*. [S.l.]: Novatec Editora, 2010. 638 p. Segunda reimpressão.

PATUCI, G. d. O. Ferramentas de teste de software. *Engenharia de Software Magazine*, 2010.

PRESSMAN, R. S. *Engenharia de Software*. 6. ed. Porto Alegre, Brasil: AMGH, 2010. 752 p. Trad. Penteado, Rosângela Ap. D.

RATIONAL, S. C. *Rational Unified Process*. 2012. Web. Visitado em 11/04/2012. Disponível em: <http://www.wthreex.com/rup/portugues/index.htm>.

RIBEIRO, C. *Penso, logo automatizo*. 2011. Web. Acessado em: 11/04/2012. Disponível em: <http://www.bugbang.com.br/?p=2108>.

SARTORELLI, R. C. Proposta de processo para automação de testes em sistemas legados. *CENTRO UNIVERSITARIO SENAC*, São Paulo, Brasil, p. 123, 2007. Trabalho de conclusão de curso - CENTRO UNIVERSITARIO SENAC.

SELENIUMHQ. *SeleniumHQ*. 2012. Web. Acessado em: 11/04/2012. Disponível em: <http://seleniumhq.org/>.

SILVA, T. X. L. d. Geração automática de scripts de teste utilizando o selenium. *Universidade de Pernambuco*, Recife, Brasil, p. 56, 2011. Trabalho de conclusão de curso - Escola Politécnica - Universidade de Pernambuco.

SIQUEIRA, H. L. d. F. Target scripts generation: um plug-in de geração automática de scripts de teste. *Universidade Federal de Pernambuco*, Recife, Brasil, p. 60, 2010. Trabalho de conclusão de curso - Centro de Informática - Universidade Federal de Pernambuco.

APÊNDICE A – Telas Prototipadas

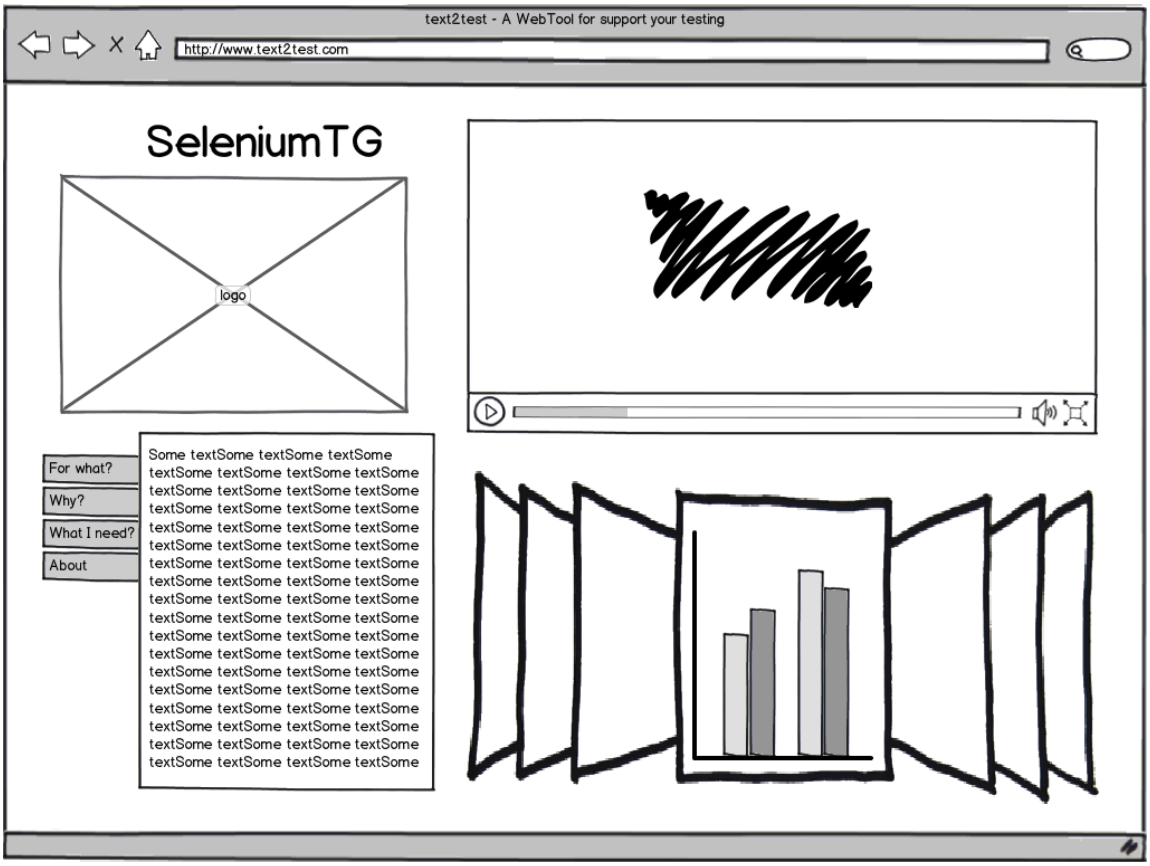

Figura 20: Protótipo da tela inicial da ferramenta.

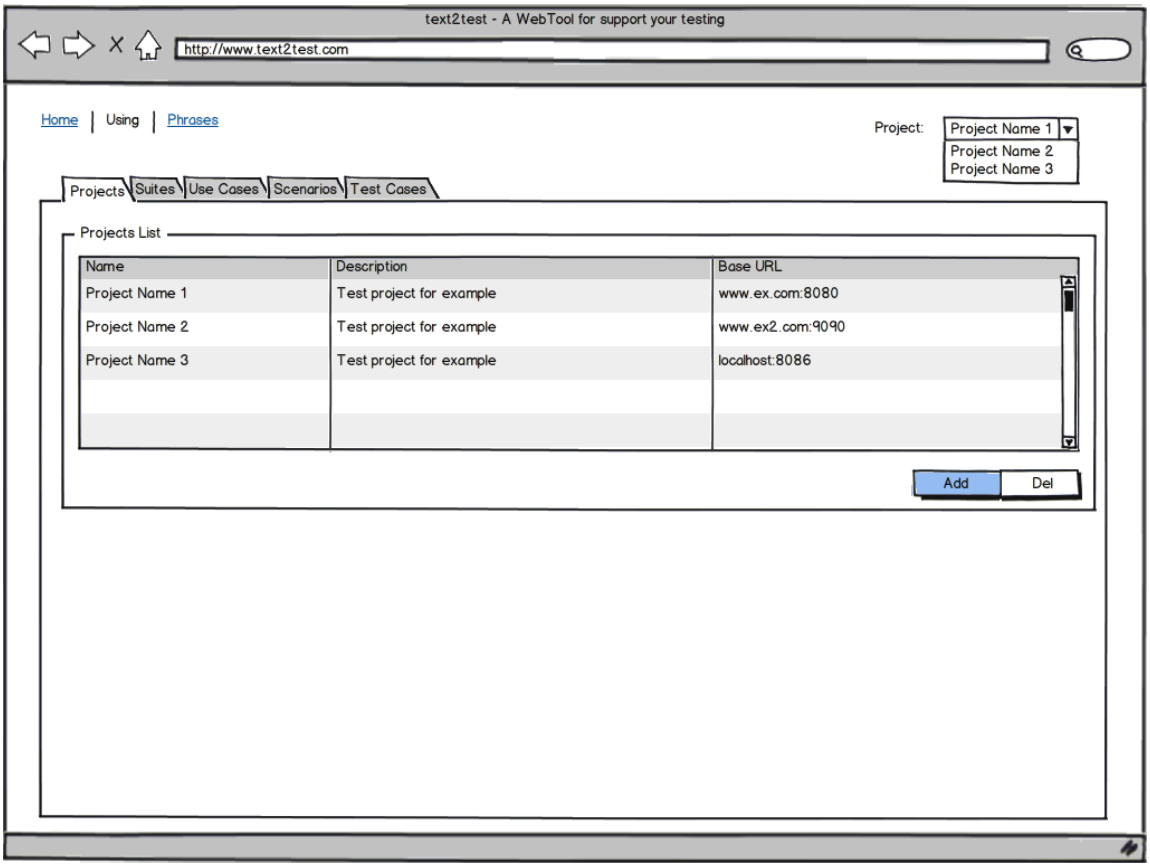

Figura 21: Protótipo da tela de manutenção dos projetos.

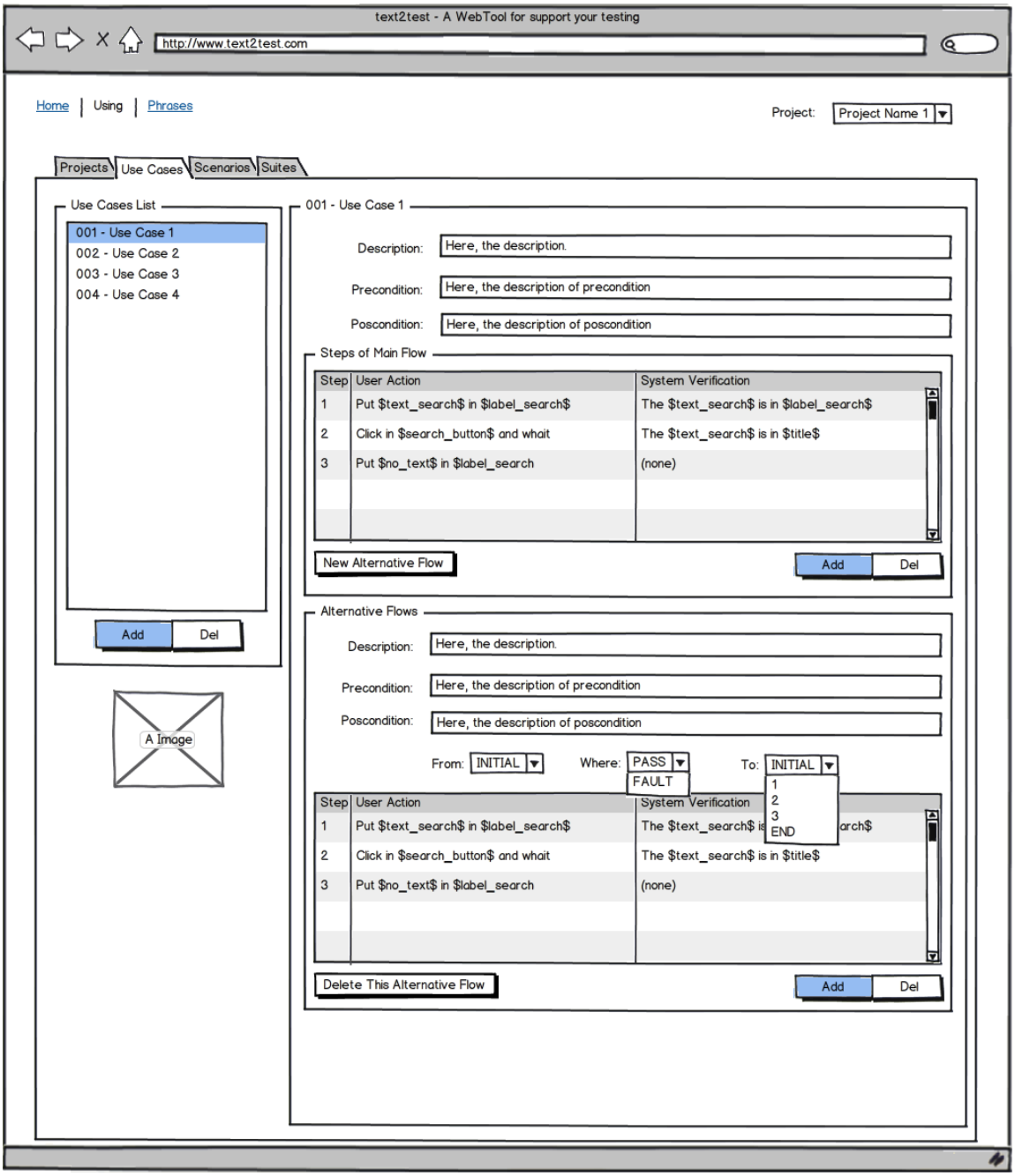

Figura 22: Protótipo da tela de manutenção dos Casos de Uso.

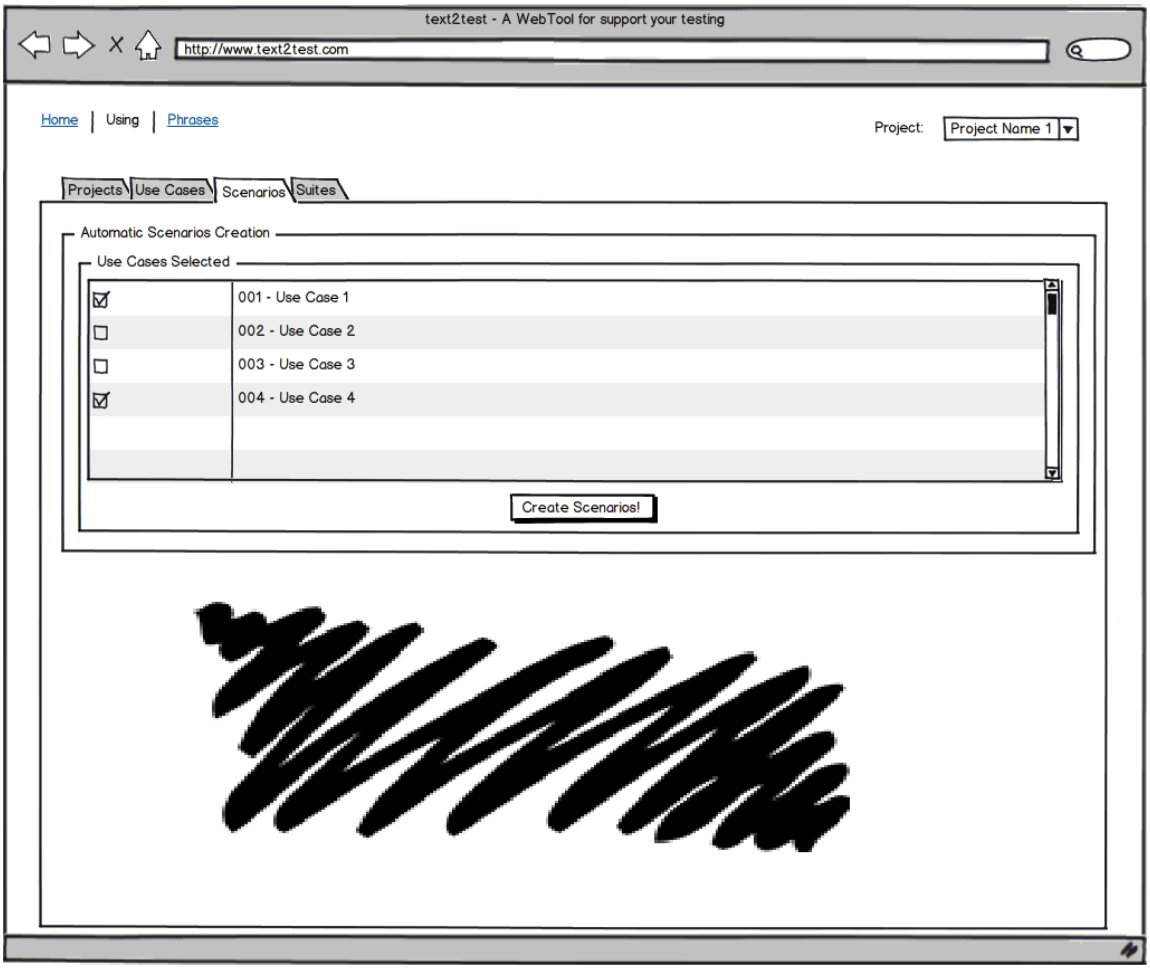

Figura 23: Protótipo da tela de criação dos cenários quando integrado com algoritmo.

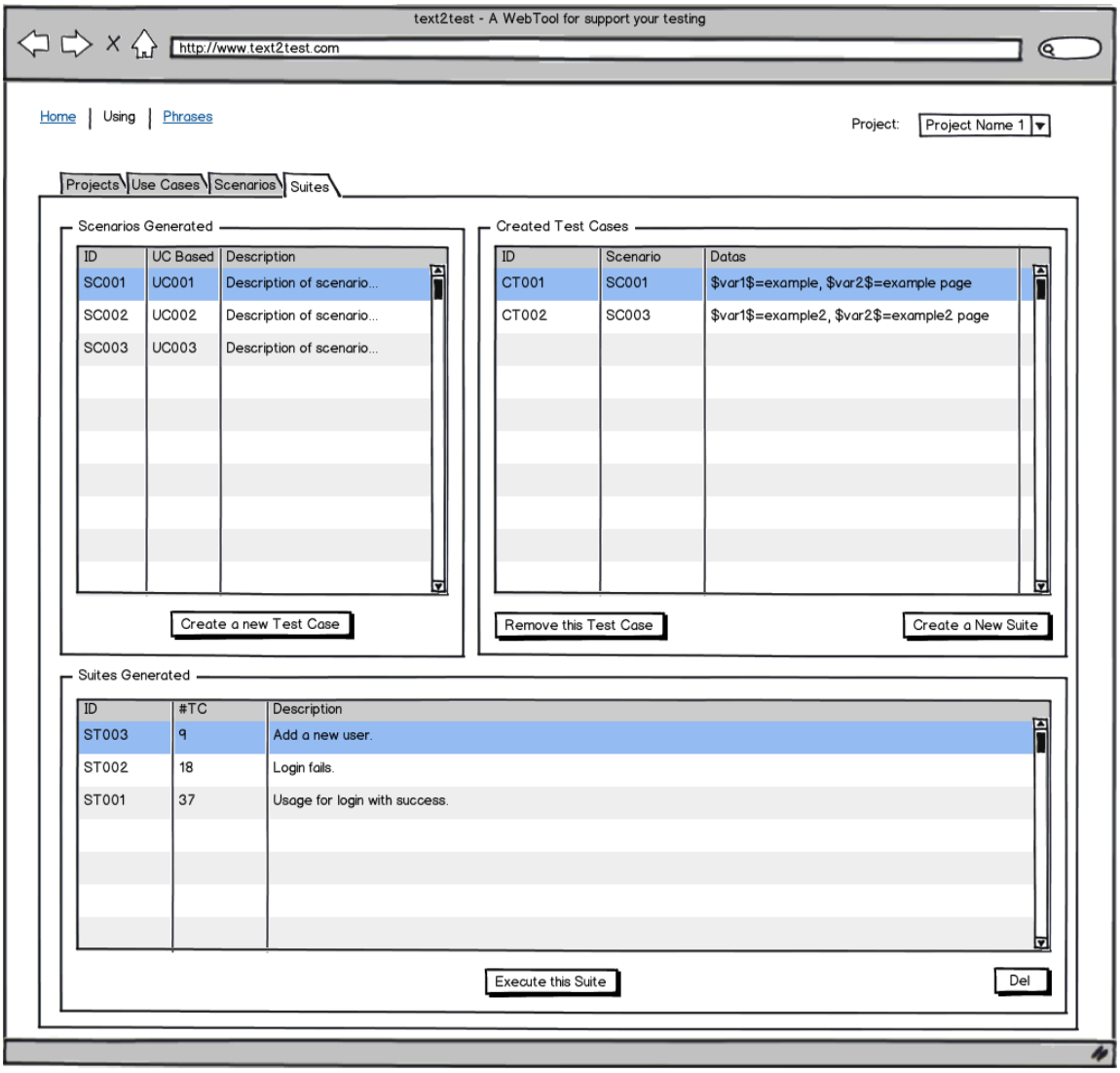

Figura 24: Protótipo da tela de criação dos casos de teste e das suítes de teste.

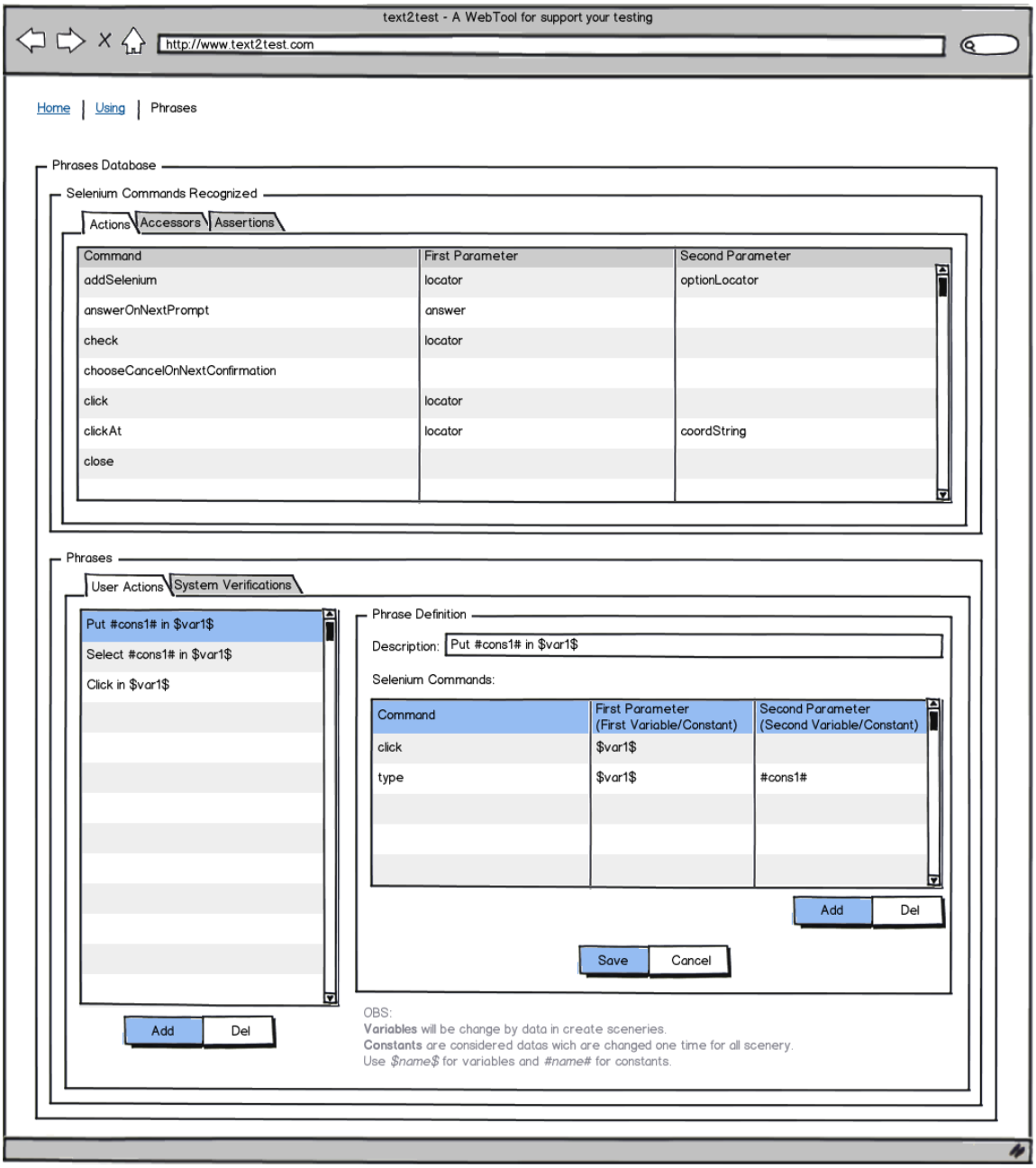

Figura 25: Protótipo da tela de manutenção das frases utilizadas nos casos de uso.