Pós-Graduação em Ciência da Computação

"Uma Domain-Specific Language para Automação de Testes de Variação de Entrada de Dados em Softwares Web no Processo de Teste em uma Empresa do Porto Digital"

Por

Francisco Salânio Vieira de Santana Júnior

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife-PE, Agosto/2012

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Francisco Salânio Vieira de Santana Júnior

"Uma Domain-Specific Language para Automação de Testes de Variação de Entrada de Dados em Softwares Web no Processo de Teste em uma Empresa do Porto Digital"

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *André Luís de Medeiros Santos*

Recife-PE, Agosto/2012

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Santana Júnior, Francisco Salânio Vieira de
Uma domain-specific language para automação de testes de variação de entrada de dados em softwares web no processo de teste em uma empresa do Porto Digital / Francisco Salânio Vieira de Santana Júnior. - Recife: O Autor, 2012.
xiv, 84 folhas: il., fig., tab.

Orientador: André Luís de Medeiros Santos.
Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2012.

Inclui bibliografia e apêndice.

1. Engenharia de software. 2. Linguagem de programação. I. Santos, André Luís de Medeiros (orientador). II. Título.

005.1 CDD (23. ed.) MEI2012 – 148

Dissertação de Mestrado apresentada por **Francisco Salanio Vieira de Santana Junior** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Uma Domain-Specific Language para Automação de Testes de Variação de Entrada de Dados em Softwares Web no Processo de Teste em uma Empresa do Porto Digital**" orientada pelo **Prof.** Andre Luis de Medeiros Santos e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Sergio Castelo Branco Soares
Centro de Informática / UFPE

_____

Prof. Clauirton de Albuquerque Siebra
Departamento de Informática /UFPB

_____

Prof. Andre Luis de Medeiros Santos
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 27 de agosto de 2012

_____

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

à minha família

Agradecimentos

Meus agradecimentos a Deus pela vida que me foi dada para que pudesse aproveitá-la da forma que me for satisfatória.

Aos meus pais por terem me dado educação e incentivo para seguir meus sonhos. Aos meus avós paternos por terem imensa influência em todos os aspectos da minha vida. À minha avó materna que tenho muito carinho e que infelizmente não está presente em vida, mas que com certeza está em grande felicidade por este momento. Aos outros membros da família que também possuem grande parcela da minha felicidade, como minhas irmãs e meus primos. Aos meus tios paternos por terem influências na minha formação como pessoa, principalmente ao meu tio Sandro Santana por ser uma referência em termos de sabedoria. Às minhas tias por serem simplesmente maravilhosas comigo e que sempre me deram incentivo e colaboram muito para o meu crescimento.

À minha esposa Taiana Santana por estar presente, mesmo quando esteve distante por motivos que a vida impõe, sempre me incentivando nos momentos difíceis e muito feliz por minhas realizações. Obrigado por tudo! Te amo, vida!

À família da minha esposa por sempre serem maravilhosos comigo.

Ao meu orientador André Santos por ser o porto seguro no que diz respeito a sabedoria e aconselhamento, me guiando pelos caminhos certos para a realização deste trabalho.

Aos professores Sérgio Soares e Renata Rodrigues de Souza por darem aconselhamentos essenciais em suas áreas em complemento ao trabalho de André Santos.

À Universidade Federal de Pernambuco que me concedeu os meios para que eu pudesse realizar o meu trabalho de forma eficaz. À FACEPE (Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco) pelo auxílio financeiro.

Aos amigos do Pará Josivan Reis, Wendeson Silva, Bruno Silva e Roberto Pereira pela motivação de partilharmos do mesmo objetivo, ultrapassando os mesmos obstáculos. Ao amigo Waldenor Jr. pela força à distância. Ao amigo Alexandre Santana pela irreverência e muita positividade dirigidas a mim. Aos novos amigos de Alagoas Felipe Buarque, Leonardo Fernandes e Fernando Kamei por terem sido grandes integrantes de grupo de trabalhos.

Às empresas Joy Street e Manifesto Game Studio por me darem a oportunidade de trabalho e que me permitiram grande flexibilidade e condições para que eu realizasse meu estudo.

Aos integrantes da banca Sérgio Soares (Universidade Federal de Pernambuco) e Clauirton de Albuquerque Siebra (Universidade Federal da Paraíba) por avaliar este trabalho.

A well-informed mind is the best security against the contagion of folly and of vice. The vacant mind is ever on the watch for relief, and ready to plunge into error, to escape from the languor of idleness

—ANN RADCLIFFE (The Mysteries of Udolpho, 1764)

Resumo

Esta dissertação apresenta uma nova ferramenta e método de automação de testes de *softwares web* com variação de entrada de dados, controle de fluxo e repetição, tendo como motivação um caso específico de uma empresa da indústria de software. Uma característica que marca o método é ser fundamentada em uma linguagem de programação específica para o domínio de teste, o que caracteriza a linguagem como uma Domain-Specific Language, que busca através de uma gramática restrita, mas expressiva para um domínio, comunicar melhor o que está sendo descrito. O trabalho mostra que houveram uma série de testes com protótipos de linguagens até chegar numa solução viável. A linguagem sofreu algumas melhorias e algumas funcionalidades foram implementadas para aumentar o potencial do método proposto. Este é uma alternativa para o método atualmente estabelecido em vários processos de teste de *software* em empresas na indústria. O método atualmente praticado possui alguns problemas que motivaram este estudo, alguns em menor escala que foram solucionados e apresentados nesta dissertação. No entanto, o problema maior, o da variação de entrada de dados para testes motivou este estudo. O objetivo é melhorar a produtividade do processo que norteia a atividade de teste em se tratando de se aplicar a variabilidade de entrada de dados nos *scripts* de teste, além de possibilitar controle de fluxo e repetição. Para avaliar a melhoria, um estudo foi executado internamente numa empresa com a equipe de teste, aliado à uma análise qualitativa através de uma discussão sobre o resultado obtido.

Palavras-chave: domain-specific languages, teste de software, software web

Abstract

This dissertation presents a new method for web software test automation with variance of input, flow control and loop support. Having as motivation, a specific case of a company in the software industry. One feature that marks the method is to be based on a specific programming language for the test field, which characterizes the language as a Domain-Specific Language, which search, through a restricted grammar, but significant for a domain, better communicate what is being described. The work shows that there were a series of tests with prototype languages to reach a viable product. The language has undergone some improvements and some features that were implemented to increase the potential of the method. This is an alternative to the method currently established in many test processes, established in various companies in the industry. The current method has some drawbacks that motivated this study, some on a smaller scale have been solved and presented in this dissertation. However, the biggest problem, the test's variation of the input data led to the study. The goal is to improve the productivity of the process that guides the test activity in the case of applying the variability of input data, flow control and loop support in test scripts. To assess the improvement, a study was performed inside a company with their test team, combined with a qualitative analysis through a discussion about the results.

Keywords: domain-specific languages, software testing, web software

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ANTLR ANother Tool for Language Recognition

DSL Domain-Specific Language

DSM Domain-Specific Modeling

EBNF Extended Backus-Naur Form

FSM Finite State Machine

GPL General-Purpose Language

GUI Graphical User Interface

LAST Linguagem de Automação de Suites de Teste

LOC Lines of Code

OJE Olimpíada de Jogos Digitais e Educacionais

TDD Test-Driven Development

1 Introdução

The first step towards getting somewhere is to decide
that you are not going to stay where you are.
—MORGAN, JOHN PIERPONT

As seções apresentadas a seguir tem como propósito introduzir esta dissertação através da discussão da motivação, objetivo e o resumo das contribuições.

1.1 Motivação

Trabalhos que necessitavam de grandes recursos de mão-de-obra, atualmente tem o suporte automatizado de sistemas de *software*. No entanto, em busca da evolução contínua, grandes investimentos são feitos em conhecimento para se aperfeiçoar os processos de produção desses sistemas.

Uma área que merece muito destaque, e é parte essencial no ciclo de produção é o teste do produto a ser gerado. A concepção do mesmo deve ter em vista a qualidade, que é demonstrada por vários aspectos, primeiramente com sua aparência, através da interface, que em muitos casos dita o sucesso de um produto/serviço.

Estatisticamente falando, de acordo com Beizer (1995), dependendo da aplicação e a qualidade do processo de desenvolvimento, as estimativas do custo de testes variam de 30% a 200% do custo de desenvolvimento. Além disso, custos de manutenção também tem uma participação bastante alta no custo total Coleman *et al.* (1994), os quais podem sofrer efeitos colaterais de um mal emprego dos testes no processo de desenvolvimento.

(Kam and Dean, 2009) menciona uma pesquisa, de um grupo americano, a qual aponta um resultado alarmante em relação a *sites* da *web* em controle do governo americano. Este estudo cita que aproximadamente 70% das aplicações continham falhas. Levando-se

em consideração esta estatística, (Li *et al.*, 2011) considera que outros *sites* em atividade na internet devam esconder falhas incontáveis, tendo-se em conta que, teoricamente, os sites de um governo como dos E.U.A. deva sofrer testes a um nível desproporcional à uma aplicação privada.

Avaliando o cenário que a disciplina de teste de *software* está situada, apresentando resultados preocupantes, como o do governo americano, e potenciais perdas milionárias em negócios feitos *online*, é natural que novos requisitos surjam para uma prática mais eficiente em testes de *software*.

Ao longo dos últimos anos, novas metodologias surgiram no intuito de se potencializar a qualidade na produção de *software*. Uma tendência que conquistou a indústria foram as metodologias ágeis. Test-Driven Development (TDD), por sua vez, é uma prática de desenvolvimento muito comum nos processos de desenvolvimento na indústria. A prática presa o teste em primeiro lugar, literalmente, antes que se escreva qualquer lógica de negócio. Portanto, houve um inegável avanço em termos de metodologias voltadas para a solução de problemas de análise de qualidade de sistemas produzidos. Muitas ferramentas foram criadas em cima de práticas como TDD, como a ferramenta de testes de aceitação Selenium, criada originalmente pela ThoughtWorks [1] (empresa muito ativa na *comunidade de programação ágil*). Esta ferramenta, relacionada em alguns estudos, e muito presente na indústria, é utilizada nesta dissertação. Na realidade, este trabalho apresenta uma técnica para suprir alguns problemas encontrados em sua utilização.

Segundo Im *et al.* (2008), testar um software envolve três fases essenciais. Primeiro, os testes são planejados. Exatamente o que testar e como é determinado. Em segundo lugar, os testes e a infraestrutura de teste são construídos. Um ambiente no qual o software sob teste será executado é montado ou construído. Finalmente, os testes são executados e os resultados são avaliados.

A motivação principal para esta dissertação consiste em atacar as duas últimas fases (Im *et al.*, 2008), no contexto de um problema detectado numa empresa situada no Porto Digital, em Recife-PE. A empresa possui um método de execução de testes que necessita de uma nova abordagem para melhorar a cobertura de testes sem que haja um entrave com a produtividade, notado na realidade atual da mesma. Partindo do estudo de caso específico, é possível que outras empresas que passam pelo mesmo problema sejam beneficiadas com a proposta deste trabalho.

A infraestrutura de testes em que este trabalho foca consiste no aparato utilizado para a realização do teste, que simplificadamente consiste na ferramenta Selenium aliada

[1] http://www.thoughtworks.com

a uma documentação de casos de teste que guiam o processo. Algumas restrições das funcionalidades da ferramenta motivaram uma abordagem que tem como cerne uma linguagem que estende a funcionalidade da infraestrutura existente, que será detalhada no Capítulo 3.

A fase final consiste no método de execução de teste. Atualmente os testes são executados com dados únicos, ou seja, os mesmos dados estáticos obtidos na criação dos testes. O método atual de execução apresenta alguns entraves, o que motivou um método alternativo que permita uma maior cobertura de testes através de uma técnica de variação de dados de entrada e controles de fluxo de execução para suítes mais elaboradas, ou seja, contendo controles condicionais e de repetição, além de um sistema de recuperação de falhas.

1.2 Objetivo

O principal objetivo desta dissertação é elaborar um novo método de testes de aceitação para a aplicações *web*, partindo de um estudo de caso em uma empresa, buscando resolver um problema prático pertinente a realidade desta, que possa ser replicado para outras. O método consiste em criar uma Domain-Specific Language (DSL) que permita estender a funcionalidade do Selenium, permitindo que se obtenha maior cobertura de testes com maior produtividade.

O método também promove, levando-se em consideração a nova ferramenta, uma adaptação do processo de teste que forneça o suporte a nova abordagem. Além disso, esta dissertação apresenta um estudo que tem como propósito caracterizar os benefícios e problemas da nova abordagem em relação ao método corrente.

1.3 Contribuições

Em resumo, as contribuições desta dissertação são:

- Método de testes de regressão, incluindo:
  - Adaptação a um processo mais ágil, que consiste em eliminar atividades repetitivas e laboriosas em detrimento a uma abordagem com características de automação;
  - Ferramenta de testes que promove uma maior cobertura de testes com maior produtividade;

- Estudo com o propósito caracterizar os benefícios e problemas da nova abordagem em relação ao método corrente.

1.4 Organização da Dissertação

Este documento está organizado em 5 outros mais capítulos além deste. O método de teste em atividade na empresa alvo de estudo de caso nesta dissertação é apresentada no Capítulo 2.

No Capítulo 3 será descrita o método proposto para suplantar características que entravam o processo de teste incluindo a implementação da DSL.

Em seguida, o Capítulo 4 relaciona o novo método proposto com o método apresentado no Capítulo 2.

O Capítulo 5 apresenta os Trabalhos Relacionados, contendo estudos, que acrescentam à área de teste de *software* através da automação dos mesmos, e também são de grande influência para este trabalho.

Por fim, no Capítulo 6, são apresentadas as conclusões e trabalhos futuros.

2

Método de Automação de Testes Atualmente Praticado na Empresa do Estudo de Caso

Este capítulo aborda a proposta que guiará o desenvolvimento da dissertação. Esta tem como objetivo a criação de uma ferramenta e método de automação de testes de formulários de aplicações *web*, contendo uma DSL para a fácil escrita de tratamento de testes e manutenção dos mesmos para o teste automatizado de aplicações *web*.

O propósito geral da DSL agregada ao ferramental é adquirir métricas, *feedback* rápido em tempo de desenvolvimento e como consequência, uma maior produtividade.

Para a análise de viabilidade do que propõe este trabalho, é necessário ter como ponto de partida uma perspectiva que possa servir de comparativo. Portanto, neste capítulo serão apresentados processos que dizem respeito a Testes Automatizados de Software para a Web, para meios de experimentação será feito um estudo observando um contexto específico.

Com o intuito de se obter resultados de maior qualidade, dois métodos foram extraídos de uma empresa, com autorização prévia. A empresa, situada no Porto Digital, em Recife-PE, iniciou uma reestruturação de processos com o propósito de gerir melhor seus recursos de forma geral. Teste de *software* foi um dos fatores de suma importância dada a proposta de pesquisa em execução, até o momento de escrita desta dissertação. Este é um cenário específico, contudo, de forma geral é possível familiarizar esta realidade a muitas outras, como é possível observar nos Trabalhos Relacionados, no Capítulo 5.

Os elementos pertencentes ao cenário, para a análise, consiste de uma equipe de teste que já passou por um processo de automação de teste antigo e um atualmente estabelecido. Os dados das duas participações foram obtidos seguindo a realização dos métodos com a

aplicação de métricas.

Na sequência deste capítulo serão apresentadas os métodos, iniciando-se na Seção 2.1, primeiramente com o Método Atual. Na Seção 2.2 será apresentado o Método Proposto Básico.

2.1 Método Atual

Este método consiste de uma evolução em relação ao teste manual de *software*. É uma alternativa muito superior em relação a um processo inteiramente manual, devido a eliminação de tarefas repetitivas e laboriosas que desperdiçam recurso que poderiam ser gastos no que realmente interessa. No entanto, posteriormente poderá ser visto que ainda há algumas lacunas a serem preenchidas neste processo, que representam questões de grande importância, visto que é impraticável automatizar todo o processo.

O método inicia-se de forma comum a qualquer tipo de aplicação, ou seja, na requisição de uma determinada aplicação por parte do cliente. Abstendo-se da parte burocrática da negociação de uma implementação de um projeto de sistema, a parte que interessa ao método de teste começa na elaboração do documento de requisitos, onde estão especificados as funcionalidades do sistema, além dos requisitos não funcionais, os quais não serão tratados pelo método proposto. A Figura 2.1 demonstra graficamente como era a configuração do processo. De forma resumida, o processo, da perspectiva de teste do sistema, ocorre da seguinte forma: o analista recebe os requisitos do cliente, elabora um documento de forma a elicitar no formato de Casos de Teste, os quais serão utilizados pelos engenheiros de teste aliado a implementação para a geração de *scripts* de teste através de uma ferramenta chamada Selenium IDE e execução destes no navegador. A seguir as etapas serão vistas detalhadamente.

2.1.1 Etapa 1

O documento de requisitos não foge à regra do que se produz na indústria, a Figura 2.2 apresenta um exemplo deste documento, que como pode ser observado, apresenta uma adaptação para casos de teste (CT: Caso de Teste), ao invés de casos de uso. Esta abordagem caracteriza-se como uma tentativa de unificação de documentos no intuito de se amenizar as dificuldades de manutenção no que diz respeito a teste de software, que como será visto posteriormente, é um problema sério, comumente presente em projetos de sistemas como o apresentado aqui. No entanto, esta abordagem apresenta um problema por abordar cenários de casos de uso de forma dispersa, não absorvidas numa visão mais

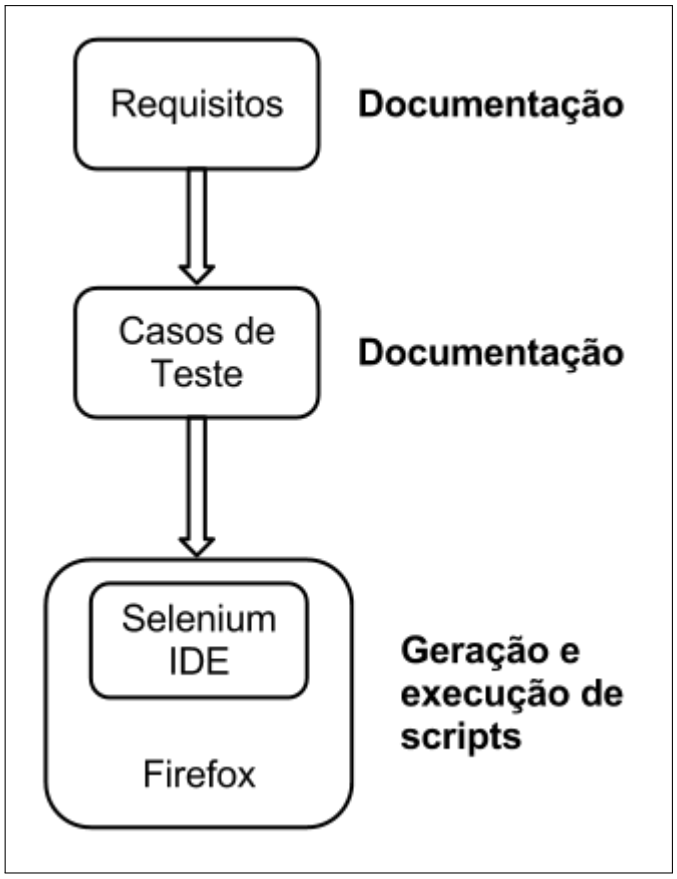

Figura 2.1 Método Atual de Automação de Testes na empresa

global de um requisito, perdendo em organização. Por outro lado, esta prática poderia ser evoluída para o desenvolvimento de uma documentação mais completa, semelhante ao que acontece com implementações que seguem o TDD (Test Drive Development)[1], tornando o seu código uma documentação viva, com exemplos de utilização.

Segundo a Figura 2.1, a etapa 1 referencia a elaboração do documento de requisitos, tarefa esta de responsabilidade do analista de negócio do sistema obtido através de interações com o cliente.

[CT002] Visualizar a página de Fale Conosco

Descrição	Passos para visualizar as dúvidas mais freqüentes
Pré-condições	Um usuário professor deve estar previamente cadastrado no sistema.

Procedimento	**Verificação**
1. Abra o navegador web e digite o endereço do sistema.	1. Será exibida a tela inicial conforme descrito no passo 1 do CT001.
2. Clique no link **Ajuda** na barra de menus.	2. Será exibida uma tela contendo uma barra de links para informações sobre Cadastro, Acesso, Torneios/Jogos, Problema Técnicos e Fale Conosco e informações relacionadas a Cadastro.
3. Clique no link Fale Conosco	3. Será exibida uma tela com uma descrição e os seguintes campos para preenchimento: - Assunto - Nome - Email - Mensagem -botão Enviar.

Figura 2.2 Exemplo de Documento de Requisitos

2.1.2 Etapa 2

Posteriormente, na etapa 2, acontece o desenvolvimento do sistema propriamente dito, levando-se em consideração, obviamente, as funcionalidades requisitadas pelo cliente. No entanto, em algumas ocasiões nem tudo é satisfeito segundo a ótica de quem requisita o sistema, portanto é necessário uma equipe que possa testar o sistema para fazer com que se cumpra o contrato de funcionalidades acordadas no documento descrito anteriormente. Pode ser o caso da contratação de uma equipe de engenheiros de teste ou capacitação da equipe presente.

Um forte aliado à esta etapa é o teste unitário, que como pôde ser visto nos Trabalhos Relacionados, é um grande auxílio para a análise de qualidade de um determinado sistema

[1]Mais informações sobre TDD em Beck (2002)

em tempo de desenvolvimento. Técnica que tem como grandes incentivadoras, práticas ágeis como o TDD, que consiste na presença intrínseca de testes ao desenvolvimento de funcionalidades. Esta configuração de teste, em tempo de desenvolvimento, denota um acumulo de perfil de um profissional que ao mesmo tempo implementa funcionalidade e também testa a mesma, que levando-se em consideração a prática do TDD, os testes, na verdade, são escritos antes de as funcionalidades existirem. Certamente que esta prática não atesta que não há necessidade de profissionais que testem o sistema, porque além deste nível de teste praticado, o teste unitário, existem várias categorias de teste que podem garantir a qualidade do sistema, e isso envolve um perfil mais específico de profissional, ou seja, um engenheiro de teste.

2.1.3 Etapa 3 e 4

A tarefa do engenheiro de teste, ou informalmente chamado de *tester* consiste de revisar os casos de teste especificados no documento da etapa 1 e elaborar *scripts* de teste na ferramenta Selenium IDE. A ferramenta é um *plugin* para o navegador Firefox[2], no qual é possível capturar interações do usuário com o sistema através da interface do mesmo. A Figura 2.3 mostra a interface da ferramenta em discussão.

A gravação da interação do usuário ocorre da seguinte forma (não entrando em grandes detalhes):

- O usuário aciona o botão de gravar;
- Interage com a aplicação clicando em botões, digitando dados em campos de texto ou selecionado opções em campos de seleção, etc;
- Desliga o botão de gravação;

Tendo efetuado a gravação, obtém-se o *script* em HTML, em formato tabular (Figura 2.4), contendo um comando (clicar, digitar, verificar texto, etc.), um alvo (campo de texto "x", botão "y", etc.) e um valor ("Nome Sobrenome"ou "Masculino", etc.). Este arquivo é o que deve ser salvo para utilização futura, ou seja, este documento em formato HTML é o *script* que será utilizado para testes futuros, como um teste de regressão ou de novas funcionalidades. É importante mencionar que esta é uma execução cíclica do processo entre as etapas 3 e 4. Isto se deve ao fato que o Selenium IDE age em conjunto com o navegador para a geração do *script* que será utilizado em um outro momento, pela

[2]Site com informações sobre o navegador Firefox: http://www.mozilla.org/firefox/

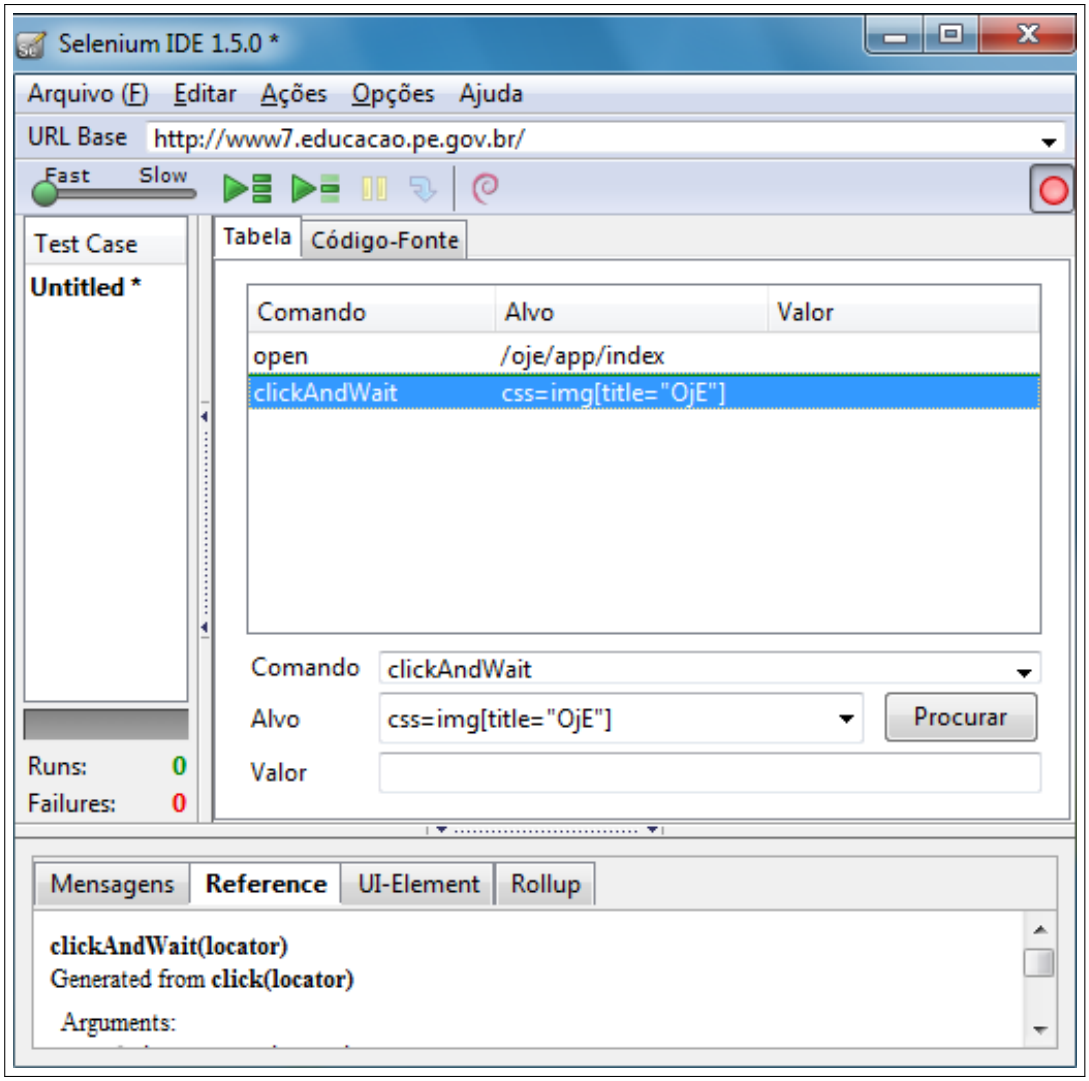

Figura 2.3 Selenium IDE

ferramenta, para execução deste *script* tendo como o alvo o navegador, ou seja, os dois *softwares* estão acoplados para a realização do propósito de execução dos testes.

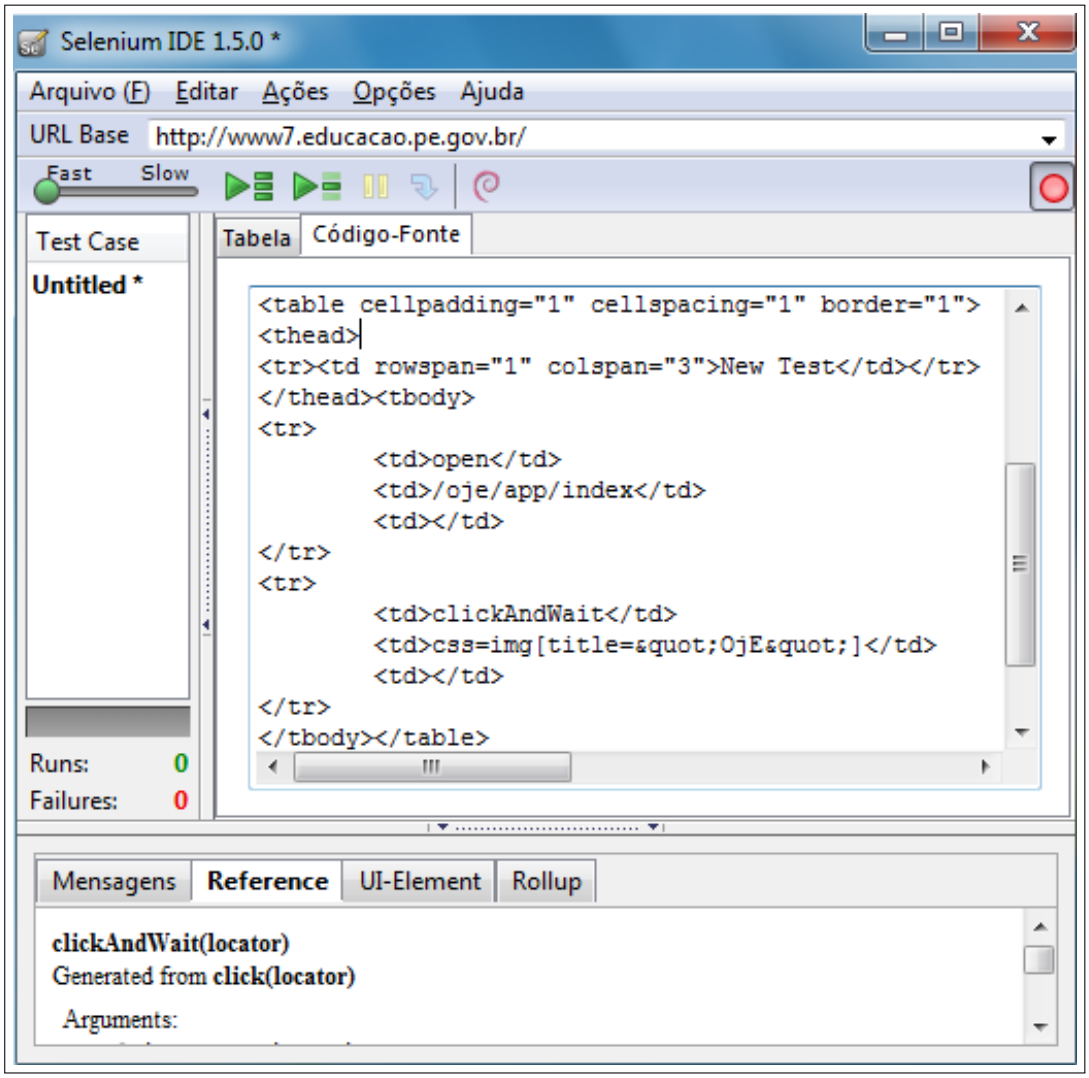

Figura 2.4 Selenium IDE: Script em HTML

2.1.4 Análise

Como comentado, este método de teste possui uma grande melhoria em relação ao teste manual, que exige um grande esforço em trabalho repetitivo que pode facilmente reproduzido por ferramentas, o que acontece neste método. A abordagem utilizou para a automação a ferramenta Selenium IDE para a gravação de interação, que gerado o *script*, este poderá ser reutilizado quantas vezes for necessário, ou seja, uma gravação de passos de um caso de uso, e várias reproduções com um clique de botão.

É um grande avanço, porém o método possui alguns problemas. Primeiramente a ferramenta testa aplicações web em somente um navegador, o que para uma plataforma tão disputada por grandes companhias disponibilizando excelentes produtos, é uma falta de análise de qualidade para um potencial grupo de usuários, ou seja, a ferramenta testa somente no navegador Firefox, deixando o grupo de usuários que utilizam o Internet Explorer, que ainda é maioria, a cargo do método manual, mais suscetível a erros que o método automatizado. Na proposta será abordada uma solução para este problema. Na realidade, e adiantando o que será discutido, a ferramenta possui suporte às outras plataformas, através de programação em uma linguagem como Java ou Ruby, o que não é interessante para este trabalho, e portanto este trabalho provê um suporte para que não haja necessidade de se ir à um nível baixo na abstração.

Outro problema pertinente a este método está na manutenção dos *scripts* de teste da aplicação em relação ao tratamento da variabilidade de *input* para cada teste além da variabilidade de fluxos de execuções. Este problema não é algo novo e muito menos específico a este cenário que esta sendo tratado nesta seção. Como pode ser visto nos Trabalhos Relacionados, vários estudos discutem uma forma de tratar o problema, e pouco de concreto surgiu além da geração de *input* aleatório, ou via uma base de dados e alguns tratam a variação de fluxos de execução com máquinas de estado.

Os pormenores da execução de um *script* de teste com variados inputs apresenta-se na seguinte forma:

- um *script* de teste existente contendo os dados de entrada do fluxo principal está no formato de tabela em HTML, tendo os dados definidos no momento da gravação dos testes;
- o *tester* executa o *script* contendo o *input* do fluxo principal;
- na intenção de se checar um fluxo alternativo qualquer é necessário alterar alguns dados de entrada. No entanto, é necessário criar uma nova gravação contendo os novos dados e novas expectativas, isso para cada variação ou então, de forma menos complicada, alterar, através da IDE os campos correspondentes aos valores diretamente;
- repete-se para cada caso de uso.

No caso da variação de fluxos de execuções, acontece da seguinte forma:

- cria-se uma suíte de teste contendo testes que caracterizam um requisito do sistema, por exemplo, cadastrar um registro num formulário, checar se o registro foi cadastrado, em uma tabela, deletar o registro da tabela, checar se o registro foi realmente excluído;
- o *tester* executa a suíte;
- na intenção de se checar uma variação de fluxo de execução que possa "quebrar"o sistema, como excluir um registro e em seguida tentar acessá-lo de alguma forma, é necessário criar uma nova suíte de testes. Caso necessite-se de executar variadas variações por uma quantidade "x"de vezes, é necessário fazê-la manualmente, criando-se cada suíte e clicando o botão de executar.
- repete-se para cada cenário.

Estes dois casos apresentados acima são os dois problemas que este trabalho ataca e que serão vistos no processo proposto. A seguir é apresentado uma parte do método proposto, o qual já foi testado pela empresa e que deverá entrar em produção num curto espaço de tempo.

2.2 Método Proposto com Funcionalidades Básicas

Recentemente houve um pequeno salto em relação ao Método Atual, visto na seção anterior. O método proposto, em estado inicial buscou resolver o problema da execução dos teste em um único navegador. Para a solução do problema foi criada uma nova ferramenta que consome os *scripts* obtidos a partir do Selenium IDE, aquele em HTML (Figura 2.4). A ferramenta foi desenvolvida pelo autor deste trabalho com todo o suporte da empresa Joy Street. Na Figura 2.5 podemos observar a configuração do processo. A seguir serão abordadas as diferenças e o funcionamento do novo método em detalhe.

2.2.1 Diferenças em relação ao Método Atual

As diferenças são a criação de uma nova ferramenta, nomeada SLang que permite a execução de testes em outros navegadores além do Firefox e com isso a alteração do processo de teste incluído a nova etapa que é a utilização desta ferramenta.

O SLang, que pode ser visualizado na Figura 2.6, é o início da ideia que culminou na proposta desta dissertação. A ferramenta surgiu com a necessidade de abranger uma maior quantidade de navegadores, nos quais a aplicação pudesse ser testada.

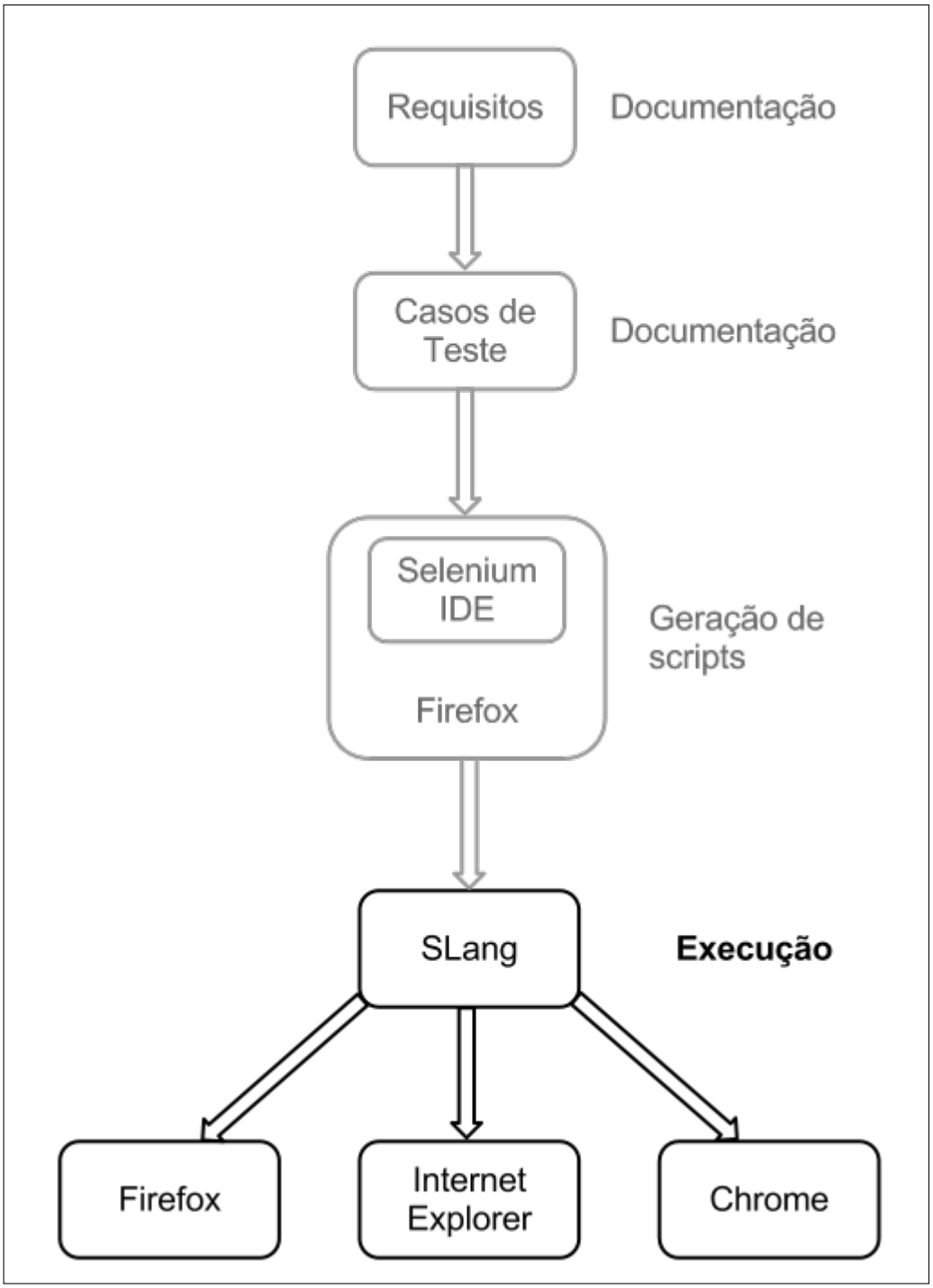

Figura 2.5 Método Proposto Básico de Automação de Teste na empresa

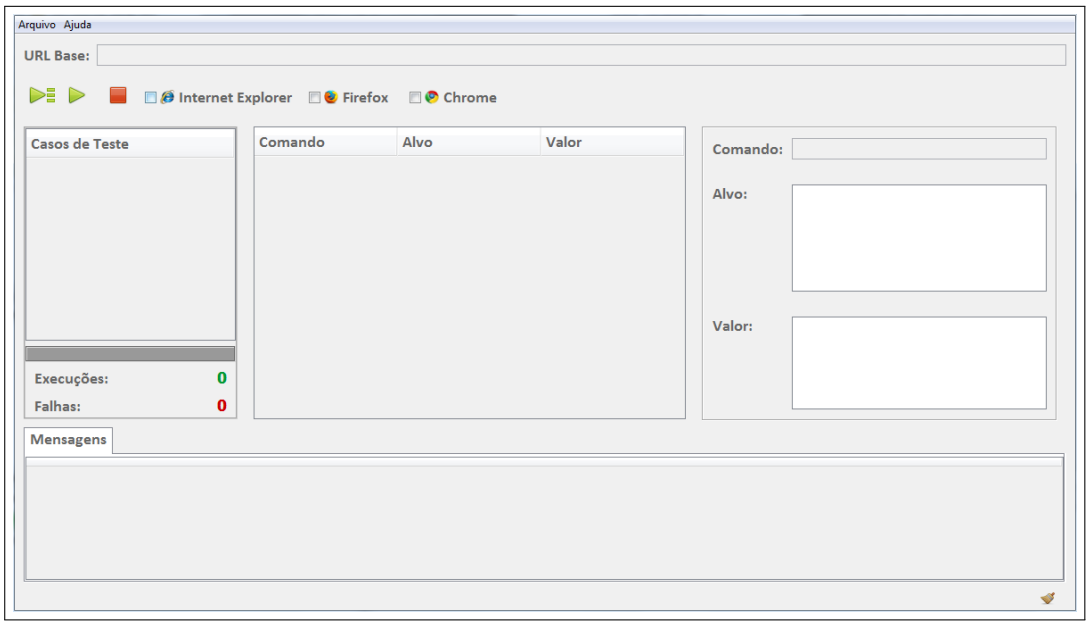

Figura 2.6 SLang: ferramenta de auxílio a execução de testes funcionais automatizados

A ferramenta SLang é baseada no Selenium 2.0 WebDriver, que é a junção de duas poderosas ferramentas, o Selenium que originalmente foi desenvolvido pela ThoughtWorks[3] e o WebDriver desenvolvido pelo Google, com o mesmo propósito: teste automatizado de aplicações web. Este Selenium mencionado é uma variação do Selenium IDE, na verdade é uma API que permite que se produza *scripts* em Linguagens de Programação de Propósito Geral (GPL) como Java, C# e Ruby que reproduz o mesmo efeito do *plugin* para Firefox, mas "programaticamente".

Na Figura 2.7 está um exemplo de *script* em Java, extraído do site[4] do Selenium. É possível notar no trecho de código da Figura 2.7 que, visualmente falando, é facilmente assimilável o que está acontecendo no código. Por exemplo, o trecho *driver.findElement(By.name("q"))* informa que será buscado um elemento pelo nome que o identifica. Esta facilidade se dá pela característica fluente da API, que é uma forma de se desenvolver DSLs internas Fowler (2011). No entanto, essa perspectiva discutida é de um praticante que possui conhecimento da linguagem Java ou de programação de forma geral com conhecimentos de Orientação a Objetos.

[3]Site da ThoughtWorks: www.thoughtworks.com
[4]http://seleniumhq.org/

```
WebDriver driver = new FirefoxDriver();

// And now use this to visit Google
driver.get("http://www.google.com");
// Alternatively the same thing can be done like this
// driver.navigate().to("http://www.google.com");

// Find the text input element by its name
WebElement element = driver.findElement(By.name("q"));

// Enter something to search for
element.sendKeys("Cheese!");

// Now submit the form. WebDriver will find the form for us from the element
element.submit();

// Check the title of the page
System.out.println("Page title is: " + driver.getTitle());
```

Figura 2.7 Trecho de um código utilizando Selenium 2.0 WebDriver

2.2.2 Análise

Foi possível observar neste método, ainda em sua incipiência, uma inclusão de um passo a mais em relação ao método atual. Só este fato já caracteriza uma desvantagem porque esse passo consiste na utilização de uma outra ferramenta e portanto adiciona mais esforço, primeiramente do conhecimento da ferramenta e posteriormente do seu uso.

O ponto positivo, porém, está no fato de se possibilitar mais testabilidade do sistema alvo, proporcionando uma maior cobertura de funcionalidades do sistema e até mesmo como requisitos não funcionais, como o de ser executado em várias plataformas, contra apenas uma do Método Atual. Portanto, a inclusão desse passo a mais no processo adiciona um maior custo de produção, porém viável devido a o resultado obtido, além de apresentar semelhanças em termos de interface em relação ao *plugin* Selenium IDE. A seguir será detalhado o processo para melhor visualizar como ocorre sua execução.

Após a execução dos passos referentes ao Método Atual (Seção 2.1) visualizável na Figura 2.1 os seguinte passos são executados para a realização dos testes:

- carregar na ferramenta SLang os *scripts*, sendo a partir de suítes de testes ou casos de teste individuais;
- selecionar navegadores, nos quais os testes serão executados e clicar no botão de executar, todos ou somente um;
- verificar erros reportados na lista de *logs* semelhante ao encontrado no Selenium

IDE.

Tendo em conta, obviamente, que este é um processo iterativo que acontece de acordo com a necessidade do usuário envolvido com a ferramenta.

Os problema deste processo em discussão são os, ainda não solucionados, problemas do tratamento de variações de *input* de dados e de fluxos de execuções, mas foi visto que houve um ganho significativo no fator de cobertura de teste de plataformas apenas com o suporte a mais delas.

2.3 Resumo

Neste capítulo foi abordado o método atual, o qual consiste do método que é atualmente praticado na empresa do estudo de caso. Este método apresentou algumas vantagens em relação à um método manual de teste, porém este contém alguns problemas, os quais são atacados no capítulo que segue.

3

Método de Automação de Testes Proposto

Este capítulo tem como propósito, apresentar o método proposto, afim de buscar solucionar algumas falhas que foram vistas no capítulo anterior. Primeiramente, pelo fato de a proposta ser fundamentada em DSLs, a seguir é apresentado uma simples introdução a respeito deste tema. Em seguida será apresentado o método proposto detalhadamente.

3.1 Domain-Specific Languages

A indústria de software busca constantemente a melhoria no processo de desenvolvimento. As DSLs tem como objetivo melhorar a expressividade e a facilidade de uso em domínios limitados em comparação às Linguagens de Programação de Propósito Geral Mernik *et al.* (2005) como Java e C#. Como exemplo de DSLs podemos ter linguagens conhecidas como CSS (Cascading Style Sheet) que tem por objetivo aplicar estilo em documentos web W3C (2012) e SQL (Structured Query Language) que é a ferramenta mais utilizada para se comunicar com um banco de dados relacional Beighley (2007). A primeira permite estilizar muito bem um documento web, como exemplo uma página HTML, no entanto você não pode utilizá-la, por exemplo, para cálculos complexos, o que caracteriza uma DSL, diferentemente de uma Linguagem de Programação de Propósito Geral, que possibilita uma gama maior de aplicações para uma variedade de domínios. É possível construir DSLs para vários tipos de domínios distintos.

Segundo Deursen and Klint (2000), Domain Specific Language é uma linguagem de programação ou uma linguagem de implementação executável, usualmente restrita a um domínio particular. Geralmente pequena, uma DSL possui característica declarativa Deursen (1998). Estes conceitos vem sendo aplicados constantemente ao longo dos anos, desenvolvendo DSLs, no entanto, o estudo sistemático vem sendo empregado apenas recentemente Fowler and Sells (2012); Deursen and Klint (2000). Atualmente, com a

identificação do conhecimento de DSL e a sua notável robustez, algumas concretizações vem surgindo ao longos dos anos na forma de aplicações, ambientes e novos ideais para esta área de muito potencial para a indústria de software a ser explorada. Algumas linguagens atualmente podem ser citadas como exemplo. Uma delas é CSS Fowler (2012), já comentada anteriormente e que tem função essencial no desenvolvimento web na última década. Há também DSLs menos populares, com a abordagem para adaptores de vídeo, presente no trabalho Thibault *et al.* (1999).

Ambientes como Excel são exemplos aproximados que Martin Fowler, em Fowler (2005) propõe como uma nova forma de se desenvolver software. Segundo Fowler (2005) Language Workbenches seria algo como a possiblidade se criar uma DSL (schema), um ambiente de edição próprio para a DSL, e geradores para esta linguagem. Este conceito consiste em um paradigma de programação chamado de Programação Orientada a Linguagem Dmitriev (2004).

DSL não é uma abordagem nova, é bastante utilizada, porém não conscientemente Fowler (2012). Fowler and Sells (2012) explica que há necessidade de se entender melhor o que são DSLs para poder melhor produzi-las. Entendendo-as, é possível afirmar que é uma linguagem de programação de pouca expressividade e focada em um domínio específico Fowler (2011). Segundo Soares *et al.* (2007) linguagens declarativas são usualmente criadas para um domínio específico, com um foco específico, o que assemelha linguagens declarativas à DSLs. Um exemplo de linguagem declarativa é a NCL, que é voltada exclusivamente para o domínio de TV Digital. Esta linguagem é baseada em XML Soares and Rodrigues (2006), porém, esta linguagem, segundo Fowler (2012), apresenta muitos ruídos em relação a uma forma mais elegante de se escrever uma linguagem compreensível por pessoas alheias ao meio do desenvolvimento de software, que pode ser proporcionada por uma DSL, aproximada da forma natural como um especialista do domínio pode compreender. Essa discussão permite apontar algumas melhores explicações das justificativas do porquê escolher DSLs:

- Aumento de Produtividade: o cerne do apelo de uma DSL é que esta provê meios para melhor comunicar a intenção de parte de um sistema Fowler (2011). Por exemplo, dada a definição de um determinado sistema, é de mais fácil entendimento se ler uma DSL que linhas de código em C++, portanto significa que é mais fácil escrever, habilitando o aumento da produção;

- Comunicação com Especialistas de Domínio: o benefício de maior impacto proporcionado pelas DSLs e de acordo com Fowler (2011) a parte mais difícil em

projetos de software. Essa qualidade permite aos especialistas no domínio entender, validar e modificar o software Deursen (1998), permitindo uma maior participação no desenvolvimento e na solução de problemas referentes ao domínio da aplicação;

- Manutenção: De acordo com os estudos de Deursen (1998) é possível visualizar benefícios além da produtividade. É possível dar manutenção em softwares fundamentados em DSLs a baixos custos.

- Custo/Benefício: No que diz respeito ao custo de usar DSLs, há evidência empírica, a qual sugere que o uso de DSLs aumenta a flexibilidade, produtividade, confiabilidade e usabilidade Deursen (1998).

Esta série de qualidades a respeito de DSLs no entanto revelam problemas inerentes a sua utilização. No entanto, DSL não resolve qualquer sorte de aplicações, porém é provável que possa justificar o seu uso para fomentar a criação de softwares de forma mais acessível a especialistas no domínio aplicado.

3.2 Apresentação do método

A Linguagem de Automação de Suites de Teste (LAST) é uma linguagem para a descrição de suítes de teste de aplicações *web*, a qual surgiu com a necessidade de elaborar suítes mais complexas de uma forma mais prática e bem específica, portanto uma DSL encaixa-se com uma alternativa mais apropriada a uma General-Purpose Language (GPL), como Java ou C#.

O porquê de se criar uma nova linguagem está no fato de facilitar o processo operacional de elaboração dos *scripts*, no qual uma GPL pode exigir um custo adicional de programação do responsável pelos testes, o que não é sempre ideal. O cenário em que surgiu a necessidade de uma técnica mais acessível demonstra este fator da programação, sendo nesse caso um fator crucial, devido à formação dos responsáveis pelos testes não ser de programação.

A proposta apresentada teve início com alguns protótipos, que tentaram, em sua origem, resolver outros problemas que são bastante relevantes mas que a princípio mostraram-se secundários em relação ao apresentado. Este que gerou a Domain-Specific Language (DSL) sofreu alguns ajustes no caminho para a maturação, havendo algumas versões que foram trabalhadas para que estivessem a um nível mais apropriado para a equipe estar confortável com a linguagem criada.

A DSL apresentada neste capítulo tem como objetivo apresentar uma gramática que permita uma baixa curva de aprendizado e rápida produtividade devido a sua característica menos expressiva do que uma GPL.

A seguir será apresentada uma breve especificação da linguagem, apresentando as suas características e funcionalidades além de como foi produzida.

3.3 Especificação da Linguagem

Esta seção está dividida em quatro subseções: Produção, que discorre sobre as decisões tomadas em relação ao desenvolvimento da linguagem; a Gramática, a qual apresenta a gramática da linguagem; em seguida é apresentado o Sistema de Tipos da linguagem e finalmente como tudo se comporta em conjunto em uma Arquitetura.

3.3.1 Produção

A produção da linguagem envolve algumas decisões sobre que ferramentas utilizar, apresentando algumas possibilidades e justificativas de escolha.

Parser Generator

Parsers tem por característica serem complexos de se construir dada a complexidade da linguagem, mas ainda mais complicados de se manter. Uma simples alteração na linguagem pode ocasionar num processo muito laborioso de se executar. Dado que há uma série de opções que proporcionam sua geração, este trabalho faz uso de um *Parser Generator*, por ser mais coerente focar nas funcionalidades que a linguagem criada apresenta.

Há uma série de *softwares* disponíveis que solucionam o problema da geração de *parsers*. Dentre os possíveis está o ANother Tool for Language Recognition (ANTLR)[1], o qual é baseado na linguagem Java, o qual serve de base para um outro *parser*, o Xtext[2]. Há também para Java o JavaCC[3]. Baseados em C e/ou C++, estão Bison[4] e Yacc[5].

O ANTLR foi a escolha devido ao fator experiência do autor deste trabalho primeiramente com a linguagem Java e segundo por ter feito experiências anteriores com o

[1] Site do ANTLR: http://www.antlr.org
[2] http://www.eclipse.org/Xtext
[3] http://javacc.java.net
[4] http://www.gnu.org/software/bison/
[5] http://dinosaur.compilertools.net/yacc

software. A documentação e a forte atividade da comunidade também foi um fator decisivo, havendo alguns livros com grande cobertura e grupos de discussão que fornecem bastante apoio.

3.3.2 Gramática

A gramática é apresentada na notação Extended Backus-Naur Form (EBNF), a qual permite elementos opcionais e repetições de elementos. ANTLR suporta a notação e esta será apresenta em etapas para maior legibilidade.

A gramática tem início com a regra *suite* que pode ser observada na EBNF 1

```
suite : 'suite' IDENT
            declaration+
            statement+
        'end' EOF
```

EBNF 1: *Suite*

Em *suite IDENT* define identificadores que iniciam com letras ou *underscore* seguido de letras ou números ou *underscore*. É possível observar que no *script* deve conter uma ou mais declarações de variáveis e um ou mais *statements*. Em EBNF 2 é apresentada *declaration*. *statements* é apresentada em dois blocos de EBNF, a primeira em EBNF 3 e a segunda em EBNF 4.

Nota-se que em *declaration* há *expression*, que poderá ser visualizada posteriormente.

```
declaration : IDENT ':=' expression ';'
```

EBNF 2: *Declaration*

A seguir são apresentadas as expressões presentes na linguagem. Por não permitir recursão a esquerda, o ANTLR necessita de uma construção encadeada como vista caracteristicamente na definição de expressões. É possível observar isso na EBNF 5.

3.3.3 Sistema de Tipos

A DSL por ser uma linguagem de *script*, apresenta tipagem dinâmica, apresentando os tipos a seguir.

```
statement : assignmentStatement | whileStatement
| runStatement | reportStatement | tryFailStatement

assignmentStatement : IDENT '=' expression ';'

whileStatement : 'while' expression
                    statement+
                 'end'

tryFailStatement : 'try'
                      statement+
                   ('fail' 'verify' STRING
                      statement+
                   )+
                   'end'

reportStatement : 'report' STRING ';'
```

EBNF 3: *Statements* 1

```
runStatement : 'run' IDENT fetching? ';'

fetching : 'fetching'
                '{'
                   queryIndexing ('.' queryIndexing)*
                '}'

queryIndexing : IDENT '[' expression ']' ':'
      elementsIndexing (',' elementsIndexing)*

elementsIndexing : IDENT '[' expression ']'
```

EBNF 4: *Statements* 2

```
value : IDENT | IDENT
        | BOOL | SCRIPT
        | SOURCE ';'

term : IDENT
      | '(' expression ')'
      | value

unary : ('+' | '-')* term

mult :  term (('*' | '/' | 'mod')* term)*

add :  mult (('+' | '-')* mult)*

relation :  add (('==' | '!=' | '<' | '<=' | '>=' | '>')*
            add)*

expression :  relation (('and' | 'or')* relation)*
```

EBNF 5: *Expressions*

Tipo *script*

O principal tipo da linguagem e mais complexo. Apresenta uma estrutura baseada no arquivo gerado pelo Selenium. Esta estrutura é uma lista de uma composição de elementos, como um registro ou objeto e apresenta estes elementos:

- Comando: a ação a ser executada pelo *driver* (*click, type, etc.* ,por exemplo);
- Alvo: o elemento alvo da ação (campo de texto para Endereço, por exemplo);
- Valor: o valor a ser inserido no alvo ("Alameda dos Anjos, 120", por exemplo).

A declaração de uma variável do tipo *script* é através de uma leitura de um *script* de um arquivo. O Código 1 demonstra uma declaração.

```
_script := @"c:\scripts\script1.sel";
```

Código 1: Tipo *script*: Declaração

Uma execução de *script* de teste sempre retorna um valor booleano, cumprindo uma expectativa ou não. Para manipular os *scripts* há a expressão *run*, apresentada

anteriormente na gramática, a qual executa o *scripts* e retorna o valor da expectativa, neste caso verdadeiro ou falso (*true* ou *false*).

Em Código 2 é possível visualizar o uso de uma variável do tipo *script* em um comando *run*.

```
run script
```

Código 2: Tipo *script*: Comando *run* simples

Tipo *source*

O tipo *source* representa uma fonte de dados para serem utilizados para variação de dados na execução de um *script*. A declaração de um *source* é demonstrada no Código 3.

```
_source := $"c:\sources\source.csv";
```

Código 3: Tipo *source*: Declaração

A utilização de uma variável do tipo *source* é feita através do comando *run* com o adendo *fetching*, como já visto na gramática. Esta utilização é demonstrada no Código 4.

```
run script fetching {source[0] : name[0]};
```

Código 4: Tipo *source*: Comando *run* com *fetching*

Tipo *int*

O tipo inteiro é utilizado principalmente para controle de fluxo com o já visto *try-fail* e *loops* como o *where*. A declaração do tipo *int* é demonstrada no Código 5.

Tipo *bool*

O tipo *bool* também é utilizado para controle de fluxo e repetições. A declaração do tipo *bool* é demonstrada no Código 6.

A seguir serão apresentados alguns usos da linguagem e o porquê são interessantes.

```
counter := 0;
```

Código 5: Tipo *int*: Declaração

```
_bool := true;
```

Código 6: Tipo *bool*: Declaração

3.3.4 Exemplos de uso

A linguagem tem como alvo solucionar alguns problemas do método atual, dos mais simples ao da complexidade de se habilitar a variabilidade de entrada de dados. Portanto, alguns trechos de código são apresentados para que seja observar a utilidade destes.

Uma das funcionalidades mais simples e muito necessária para a variação é executar uma suíte de testes por tantas vezes automaticamente. Infelizmente, o Selenium IDE não permite que se faça isso sem que haja interação, como o clicar do mouse para iniciar a ação. Na Domain-Specific Language (DSL) proposta é possível fazer como no Código 7, que executa um *script* 10 vezes seguidas.

```
suite exec_many
  _script := @"c:\scripts\script1.sel";
  _times := 0;

  while (_times < 5)
     run _script;
     _times = _times + 1;
  end

end
```

Código 7: Exemplo: Executar um *script* 5 vezes com variação de dados.

No caso mais comum para o teste apresentado no Código 7, seria habilitar a variabilidade. Para isso é necessário fazer uso do tipo *source* como apresentado anteriormente. O Código 8 demonstra este uso.

Outra adição em relação ao método atual, é a funcionalidade de recuperação de falha, permitindo que o erro ocorrido seja reportado e permitir que o teste seja executado novamente ou até mesmo executar um outro teste. Isto é possível com o bloco *try-fail*. O exemplo de Código 9 demonstra a situação de executar um outro *script* caso o primeiro dê algum tipo de falha.

```
suite exec_many_source
  _source := $"c:\sources\source.csv";
  _script := @"c:\scripts\script1.sel";
  _times := 0;

  while (_times < 10)
     run _script fetching {
          source[_times] : name[0], login[0], email[0],
              pass[0], pass_conf[0].
          source[_times+1] : name[1], login[1], email[1],
              pass[1], pass_conf[1]} ;
     _times = _times + 2;
  end

end
```

Código 8: Exemplo: Executar um *script* 5 vezes com variação de dados de entrada.

```
suite try_fail
  _script1 := @"c:\scripts\script1.sel";
  _script2 := @"c:\scripts\script2.sel";

  try
      run _script1;
  fail
      report "Falha no script1. Executar script2.";
      run _script2;
  end
end
```

Código 9: Exemplo: Demonstração de *try-fail*

3.3.5 Arquitetura

Esta seção explica o funcionamento da solução, tanto da interação do usuário, quanto algumas abstrações da implementação.

Interação do Usuário

A atividade básica do usuário, geralmente o engenheiro de teste, consiste em elaborar um *script* na DSL proposta, tendo como base uma interação anterior com a ferramenta Selenium IDE, onde há a geração do caso de teste, já comentado no Capítulo 2. Na criação do *script* deve-se apontar para o(s) caso(s) de teste anteriormente gerados, através de declarações de *scripts* de teste, como visto na gramática. Tendo o *script* criado, o usuário deve iniciar o uso da linguagem através de uma aplicação de linha de comando passando como parâmetro o *script*. No momento da execução, é reportado algumas informações como o que foi executado em determinado instante e se houve algum erro. A Figura 3.1 demonstra a interação.

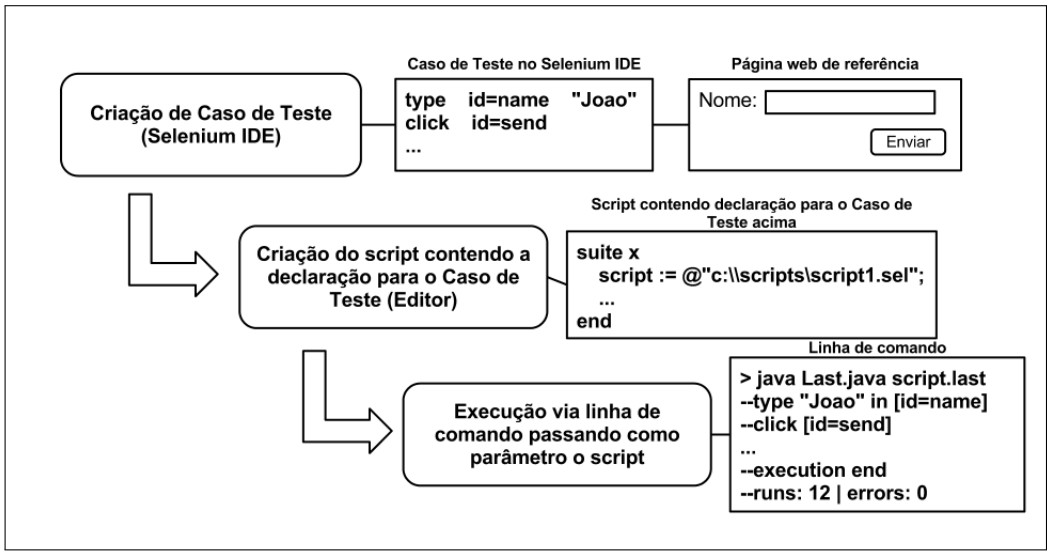

Figura 3.1 Interação do usuário com a DSL. Fonte: do autor.

Implementação

O primeiro passo para o desenvolvimento da DSL proposta foi desenvolver a gramática da linguagem, visto na Seção 3.3.2. Nesse momento é onde são definidas construções que consolidam o perfil da linguagem. No caso da solução proposta, o propósito é facilitar o trabalho de engenheiro de testes através de uma ferramenta que contenham um aspecto

que esteja inserido no contexto de seu trabalho. Com isso a DSL foi produzida com sentenças que pudessem promover a fácil assimilação da mesma por parte dos usuários.

Como segundo passo, após ter as construções que definem a sintaxe da linguagem, ou seja, a estrutura das sentenças, é necessário que haja uma forma de se interpretar as construções de uma forma compreensível, ou seja, que haja um valor semântico para que haja a interpretação. Portanto, é necessário que exista um modelo semântico, que consiste em implementar a semântica do que será produzido nos *DSL* dentro do domínio da linguagem.

Com base na sintaxe, é necessário que ações sejam tomadas, fundamentada na semântica. Simplesmente, é necessário que haja funções que dão início a alguma ação, dada uma construção dentro do contexto da linguagem. Uma forma de resolver este problema é através do padrão de *design* Interpreter. Este padrão específica como avaliar as sentenças na linguagem. De forma resumida, a idéia é que para cada símbolo que constitui a linguagem, exista uma classe (no caso, em Java). Para visualizar melhor a Figura 3.2 contém a definição do padrão.

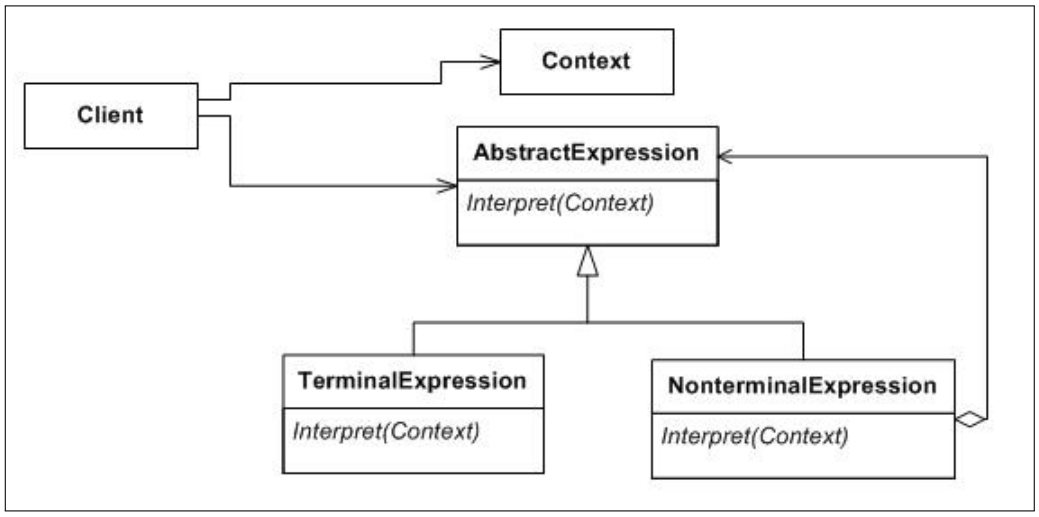

Figura 3.2 Estrutura do Padrão Interpreter. Fonte: http://en.wikipedia.org/wiki/File:Interpreter_UML_class_diagram.jpg (Creative Commons Attribution-ShareAlike 3.0 License)

3.4 Resumo

O capítulo apresentou o método proposto para tentar melhorar alguns pontos negativos apresentados no Capítulo 2. Foi visto que o método é baseado numa DSL e que apresenta

uma alternativa com grande potencial. O método proposto é confrontado com o método atual no próximo capítulo.

4 Análise

É importante que se tenha um estudo formal que produza dados que possam ser derivados em informação que avaliem um processo ou método em relação a um outro. Portanto este capítulo apresenta um estudo experimental com o método de teste proposto no Capítulo 3. Este estudo analisa uma abordagem alternativa de testes de aceitação para aplicações *web* numa empresa, avaliando sua efetividade. Este capítulo segue o formato de Soares (2004), destacando que o autor seguiu um outro *design* com objeto de controle, o que no caso deste estudo são utilizados dois tratamentos.

As seções a seguir descrevem detalhadamente o *design* do experimento e suas atividades, além dos resultados obtidos e discussões destes afim de avaliar se o método proposto permite uma melhor produtividade por parte da equipe de testes em relação ao método atualmente estabelecido.

4.1 Objetivos

As seções a seguir descrevem o objetivo deste estudo experimental em termos de Objetivo Geral, Objetivo do estudo e Métricas.

4.1.1 Objetivo Geral

Avaliar o método de teste proposto no contexto de uma aplicação *web* fazendo uso da ferramenta LAST.

4.1.2 Objetivo do Estudo

Analisar o método proposto para a realização de testes de aceitação para aplicações *web*, com o propósito de qualificar o impacto da adoção do método proposto, no que diz respeito

ao tempo necessário para construir uma suíte de testes do ponto de vista do engenheiro de *software* no contexto de uma empresa situada no Porto Digital, com engenheiros de teste, realizando atividades comumente praticadas no dia-a-dia da empresa.

4.1.3 Questão

O tempo para construir uma suíte de testes utilizando o método proposto é diferente do método corrente na empresa no contexto de teste de aceitação de aplicações *web*?

Para deixar claro, a diferença é sobre o tempo, a suíte de teste ao fim da construção é a mesma para ambos os métodos.

4.1.4 Métrica

Dados, no que se refere ao tempo gasto para se construir uma suíte de teste em um cenário de teste selecionado. A unidade dos dados utilizada é tempo em minutos.

4.2 Planejamento

Está seção apresenta o planejamento do estudo, descrevendo como o estudo foi definido.

4.2.1 Hipóteses

Antes de apresentar as hipóteses faz-se necessário apresentar um símbolo utilizado ao longo do capítulo que caracteriza a métrica a ser coletada e analisada.

TCS — Tempo de criação de suíte de testes;

Para cada métrica há uma variação, uma utilizando o método proposto. Por exemplo, o tempo de criação de suítes de testes para o método proposto tem o símbolo $TCS_{proposto}$ e o mesmo tempo para o método atual tem o símbolo TCS_{atual}.

A hipótese principal é a hipótese nula que afirma não haver diferença entre usar ou não o método proposto em relação ao método atual. Portanto, o estudo tenta rejeitar esta hipótese.

Hipótese nula (H0)

O tempo necessário para se desenvolver uma suíte de testes utilizando o método proposto não é diferente se desenvolver usando o método em vigor na empresa.

$$H0: TCS_{proposto} \simeq TCS_{atual}$$

Hipótese alternativa (H1)

Outra hipótese a ser considerada no caso de a hipótese nula ser rejeitada é a hipótese alternativa, a qual neste contexto significa: o tempo necessário para se desenvolver uma suíte de testes utilizando o método proposto é menor do que se desenvolver usando o método em vigor na empresa.

$$H1: TCS_{proposto} < TCS_{atual}$$

4.2.2 Tratamento

O estudo contém dois tratamentos. Um consiste do método proposto no Capítulo 3, o qual na prática significa um processo mais automatizado e descritivo de construção de testes de aceitação no contexto de aplicações *web*. O outro tratamento significa a utilização do método em vigor na empresa, descrito no Capítulo 2. No intuito de se identificar os efeitos da utilização do método proposto, cada participante realiza atividades para cada tratamento.

4.2.3 Objeto experimental

O estudo faz uso de um cenário funcional de um *software* real, o qual caracteriza-se como uma rede social com propósitos educacionais, voltada para estudantes de ensino fundamental e médio do sistema público bem como seus professores. O cenário escolhido para a análise, foi um cenário que possui similaridade funcional, ou seja, trata-se basicamente de formulários de cadastro contendo uma mesma quantidade de elementos gráficos, que representam a aplicação real abstraindo, porém, *designs* gráficos, que possam corroborar para um viés, tendo em vista que os participantes já realizaram testes como o método em atividade na empresa em outras ocasiões. Na Figura 4.1 é possível visualizar um exemplo.

O cenário é o Cadastro de Aluno, que é caracterizado por ser um formulário *web*, requisitando dados pessoais e acadêmicos do aluno que irá fazer parte do site. Para o cenário há um *script* de teste criado para o método de teste atual e o proposto. Além deste cenário, um outro serviu de piloto como pode ser visto no Apêndice E e que faz parte da preparação, explicada na Seção 4.2.8. É importante deixar claro que a escolha

do cenário foi aleatória, sorteando-se os mesmos. Foram colocados papéis, contendo os nomes dos dois cenários, em um saco. O primeiro sorteado foi o piloto, que pode ser visto no Apêndice E e o que sobrou foi utilizado no experimento.

Figura 4.1 Exemplo de Formulário do Cenário do Experimento.

4.2.4 Sujeitos do experimento

Os participantes do estudo são engenheiros de teste/pesquisadores da empresa, que tem como atividade a criação de testes seguindo o método em vigor, elaborando *scripts* na ferramenta Selenium, dentre outras atividades de pesquisa relacionada ao desenvolvimento de *software*, no âmbito de jogos digitais. Os mesmos passaram por um treinamento que teve como propósito prepará-los para o experimento. Este treinamento será melhor detalhando na Seção 4.2.8.

4.2.5 Variáveis independentes

- Aplicação *web* previamente criada, com base na linguagem Java, sendo o domínio Redes Sociais educacionais;
- Documento de caso de teste contendo o cenário de teste;
- Ambiente de criação de *scripts*: Selenium IDE, o qual é um *plugin* do navegador Firefox.

4.2.6 Variável dependente

- O tempo necessário para se construir uma suíte de testes;

4.2.7 Design do experimento

O experimento consiste na utilização dos dois métodos, o atual e o proposto em um cenário de teste da aplicação discutidos em 4.2.3, que ocorrerá após a preparação, apresentada na subseção seguinte.

A análise dos dados obtidos pelo experimento será feita seguindo o *design* Single Blocking Variable (Juristo and Moreno, 2001). O *design* tem como objetivo anular fatores não desejados que influenciem no resultado obtido no experimento. Para conseguir resultados que minimizem influencias de fatores não desejados, o experimento apresenta a estrutura observável na Tabela 4.1. Sendo TA e TP, respectivamente Tratamento Atual e Tratamento Proposto. Para esclarecer, os tratamentos são avaliados no mesmo cenário. Não confundir os tratamentos como sendo tipos de cenários na tabela.

Tabela 4.1 *Design* do Experimento: Single Blocking Variable

	Cenário	
Participante 1	TA	TP
Participante 2	TP	TA
Participante 3	TA	TP
Participante 4	TP	TA
Participante 5	TA	TP
Participante 6	TP	TA

Os participantes receberão como entrada para o experimento, os *scripts* de teste para cada cenário (os mesmos para os dois tratamentos) nos quais serão realizados os testes, são apresentados no Apêndice A. Um arquivo CSV contendo a variação dos dados (Apêndice B a serem inseridas seguindo a *checklist* presente no Apêndice C.

É parte das atividades referentes ao experimento, a persistência das informações ao longo do processo, ou seja, os participantes deverão guardar os dados obtidos para a métrica TCS, apresentadas na Subseção 4.2.1. A planilha que será utilizada tem uma representação que pode ser observada no Apêndice D na Seção D.1.

A execução do experimento, seguindo o design, consiste em fazer com que cada participante execute o experimento seguindo uma determinada ordem, ou seja, o Participante

5 realizou as atividades utilizando, primeiramente, o método atual e em seguida com o método proposto, diferentemente do Participante 6 que fez o inverso. A definição das ordens deu-se através de um sorteio, onde o nome de cada participante foi colocado em um pedaço de papel e colocados todos em um saco e o os dois tratamentos, da mesma forma, foram colocados em outro saco, então escolhia-se um papel contendo o nome de um participante e um outro do método, sendo o método sorteado o primeiro a ser executado e outro, portanto, o segundo. Em seguida, ao sortear um outro participante, este deveria começar pelo segundo método do participante anterior, e assim sucessivamente até que todos sejam sorteados.

Ao final do experimento é obtida uma suíte de testes para cada tratamento, contendo a variação de dados. É possível visualizar dois exemplos de suítes, um para o método atual e um para o método proposto no Apêndice F.

Os participantes também devem responder a um questionário, presente no Apêndice D na Seção D.2, cujo resultado será discutido na Subseção 4.2.11.

4.2.8 Preparação: treinamento

A preparação para o experimento consiste em se fazer um treinamento do novo método de testes de aceitação proposto. Em consequência de a ferramenta utilizada ser uma DSL e utilizar-se linha de comando para executá-la, faz-se necessário, ao menos, um apanhado geral de sintaxe e semântica da linguagem, demonstrando através de alguns exemplos práticos de forma construtiva e por partes separadas, sem que seja inserido um viés, como demonstrando através de um cenário semelhante ao que será feito no experimento propriamente dito.

Também será feito um piloto com os participantes com um cenário contendo as mesmas características do cenário presentes no experimento. Este cenário será escolhido de forma aleatória entre os dois cenários criados para o experimento.

A característica do piloto é a mesma do experimento, ou seja, o cenário aleatoriamente escolhido será repassado aos participantes que deverão realizá-los utilizando os dois métodos. O cenário piloto é apresentado no Apêndice E.

4.2.9 Análise

A análise do estudo consiste em comparar os dados coletados do experimento dos dois tratamentos aplicados no estudo com objetivo de observar se a hipótese nula pode ser rejeitada.

O estudo analisa o efeito do tratamento proposto no:

- Tempo de criação de suítes de teste;

Devido aos fatores do *design* do experimento e a quantidade de amostras pequena, foi feita uma análise estatística para descobrir se os dados seguiam uma distribuição normal utilizando o método de Lilliefors (Razali and Wah, 2011) e por fim a utilização de um outro método para descobrir se o resultado obtido rejeita a hipótese nula. Os métodos utilizados são apresentados a seguir.

Teste de Normalidade de Lilliefors

O teste de normalidade de Lilliefors é uma modificação do teste Kolmogorov-Smirnov (KS), sendo este apropriado em uma situação em que todos os parâmetros são completamente conhecidos (Razali and Wah, 2011), ou seja, o valor esperado e a variância devem ser conhecidas antecipadamente.

No caso deste trabalho, optou-se por utilizar o Lilliefors por não ter conhecimento antecipado dos parâmetros da distribuição, o que é mais conveniente.

O resultado que determina a significância do teste estatístico chama-se estatística D, e é definido abaixo:

$$D = max|F(x) - S(x)|$$

Onde S(x) é a distribuição da amostra cumulativa e F(x) é a probabilidade cumulativa normal. Se a hipótese nula é verdadeira, a amostra e as probabilidades cumulativas normais devem ser semelhantes. S(x) é definida como a proporção de valores de amostra que têm valores menores ou mesmo do que os valores de x (Razali and Wah, 2011). Se D for maior que o valor crítico presente na tabela de Lilliefors então a hipótese nula é rejeitada, o que vai indicar que os dados não seguem uma distribuição.

Análise de Variância (ANOVA)

De forma simples, e diretamente relacionada com este estudo, a Análise de Variância proporciona um teste estatístico que determina a diferença ou igualdade de médias entre dois grupos de dados ou mais. As hipóteses nula (H_0) e alternativa (H_1) da ANOVA seguem abaixo:

$$H_0 : \mu_1 = {}_2 = \ldots = \mu_{\mathrm{k}}$$

$$H_1 : H_0 = falso$$

As nula (H_0) mapeia a hipótese nula definida para o experimento que diz que os métodos propostos possuem Tempo de Criação de Script semelhantes e que dessa forma um não apresenta qualidades maiores que outro, ou simplesmente um não é melhor que o outro. Portanto, para se rejeitar a hipótese deste estudo é necessário que a hipótese alternativa da ANOVA se confirme, ou seja, que a média de TCS do método proposto seja diferente do método atual, e ainda melhor, que a média do primeiro seja maior que a do segundo.

4.2.10 Ameaças à validade

Esta seção tem como propósito discutir o quão válidos são os resultados obtidos e se é possível generalizá-los para uma população mais ampla (Soares, 2004). Existem quatro tipos de validade. Segundo Trochim (2006) a validade de conclusão é o grau de aceitação da conclusão em relação aos dados obtidos. A validade interna define se o resultado obtido é originado do tratamento e não de causas alternativas. A validade de construto preocupa-se com o certeza de que o tratamento reflete a causa e os resultados os efeitos. E por fim, a validade externa consiste na habilidade de se generalizar o estudo para outros escopos externos ao estudo.

Validade de Conclusão

A execução do estudo foi acompanhada de perto, para que não houvesse problemas com procedimentos de execução e coleta de dados. O estudo realizou testes estatísticos condizentes com o contexto dos dados obtidos, para que não houvesse influência forte de fatores externos ao tratamento.

Validade Interna

A observação do estudo pode ter tido uma causa alternativa, mais provavelmente sendo a experiência. No entanto, o *design* utilizado, que consiste em dois tratamentos no contexto de blocos aleatoriamente distribuídos, tem como característica eliminar viés dos resultados, permitindo uma observação mais nítida.

Todos os participantes tiveram contato com os dois tratamentos, sendo que cada participante aplicava-os em ordens distintas, aliviando um provável fator psicológico que possa ter causado ruído na execução do experimento.

Um fator positivo em relação ao tratamento proposto é que os participantes já possuem conhecimento do método atual, o que dá um viés contra o tratamento proposto no estudo, o que pode fortalecer ainda mais o resultado, positivamente ou negativamente.

Validade de Construto

A operacionalização do experimento é detalhada na Seção 4.2.11, mas foi possível observar em seções anteriores que seguiu-se um guia de execução para que as métricas fossem aplicadas de forma correta. Os participantes passaram por uma preparação para o experimento para ter conhecimento dos procedimentos a serem realizados, tendo-se cuidado que estes não representassem guias exatos de como realizar o experimento. Além disso as métricas foram aplicadas em cenários reais e os dados coletados buscando reportar os dados de forma relevante para o objetivo do trabalho de resolver o problema prático de uma determinada empresa.

Validade Externa

Um dos objetivos do estudo foi produzir um novo método de automação de testes de aplicações *web* que possibilitasse uma maior ganho em produtividade para uma empresa específica. O *design* do experimento utilizado buscou reduzir a conhecida variação de experiência por parte dos participantes. No entanto, devido a baixa quantidade de participantes, não é possível generalizar para outros escopos, diferentes ao especificado neste estudo, ou seja, é necessário que haja uma configuração de métodos e processos de teste muito semelhante em uma outra empresa na indústria, porém para empresas maiores não cabe, certamente, uma generalização.

Os aspectos observáveis no método estudado são bastante presentes na indústria e poder avaliá-los com outra alternativa, foi bastante suficiente para o propósito deste trabalho de forma geral.

Apesar da pequena dimensão do estudo, foi possível observar dados esclarecedores do método em vigor de uma empresa e avaliar seu desempenho, tendo como referência um outro método proposto neste trabalho, visando resolver um problema prático desta empresa.

4.2.11 Execução

Como foi discutido anteriormente, o experimento foi realizado com a equipe de testes de uma empresa voltado para um contexto real pelo o que a empresa passa no momento.

Antes da execução do experimento propriamente dito, foi feita uma reapresentação do método atual, o qual é um dos tratamentos no experimento. Reapresentado, pois a equipe já possui conhecimento do método. A reapresentação buscou informar as atividades realizadas necessárias para a criação de *scripts* e a execução dos mesmos. No caso do método proposto, o outro tratamento em estudo, foi feito um treinamento básico apresentando a sintaxe da linguagem, as suas possíveis operações e exemplos, tomando-se cuidado para não prejudicar o experimento.

Após a preparação os ambientes para os métodos foram apresentados no estado a serem utilizados, explicando-os aliado ao *checklist* apresentado no Apêndice C. Para tal, foi realizado um piloto com um formulário *web* contendo elementos gráficos semelhantes, porém com ordenação distinta do cenário utilizado no experimento. Com isso os participantes puderam se familiarizar com o procedimento, tomando cuidado com a planilha de coleta de dados e com o correto controle do seu tempo para que o experimento fosse concluído sem grandes problemas.

A seguir serão apresentados os dados coletados com a condução do experimento que ocorreu no ambiente da empresa.

Dados do Estudo

A Tabela 4.2 apresenta os tempos de criação de *script* para o método atual em minutos, sendo a fração os segundos, por exemplo, 24,133333 minutos são 24 minutos e 8 segundos. A Tabela 4.3 apresenta os dados para o método proposto.

A Tabela 4.4 apresenta as médias de criação dos *scripts* de ambos os métodos para uma verificação a ser feita na análise estatística na próxima seção.

Tabela 4.2 Dados: Tempo de Criação de Suíte (TCS) para o Método Atual.

Participante	Tempo de Criação de Suíte (TCS)
Participante 1	24,133333
Participante 2	10,100000
Participante 3	11,466667
Participante 4	27,100000
Participante 5	16,833333
Participante 6	16,383333

Tabela 4.3 Dados: Tempo de Criação de Suíte (TCS) para o Método Proposto.

Participante	Tempo de Criação de Suíte (TCS)
Participante 1	7,4500000
Participante 2	4,6166670
Participante 3	8,1166670
Participante 4	8,1500000
Participante 5	7,7166670
Participante 6	7,3666670

Tabela 4.4 Médias para o Tempo de Criação de Suíte (TCS) para o Método Atual e Proposto.

	Média TCS (min)
Método Atual	17,6694443
Método Proposto	7,2361113

Análise Estatística

Como previamente discutido, para decidir que tipo de teste estatístico utilizar, foi necessário testar se os dados seguiam uma distribuição normal. Portanto, o teste de normalidade de Lilliefors foi utilizado para testar a normalidade dos dados, ou seja, se os dados seguem uma distribuição normal. Para isso é necessário que a hipótese nula (dados seguem distribuição normal), vista na Seção 4.2.9, não seja rejeitada.

É importante deixar claro que o tempo de execução não foi avaliado pois os dados observados em outras ocasiões não apresentou influência considerável para realizar o teste estatístico, portanto apenas o TCS será analisado seguindo o modelo estatístico.

Com a ajuda da ferramenta R[1] é possível realizar o teste de Lilliefors de forma simples. Como os dados obtidos por testes feitos com o mesmo grupo, as amostras são dependentes, porque cada participante realizou os dois métodos, estes puderam ser aglomerados num vetor que pôde ser passado para o método estatístico como entrada.

A execução do teste de normalidade retornou como saída dois valores:

D = 0,226

p-value = 0,09142

De acordo com o que foi visto na Seção 4.2.9, o valor crítico para o valor D, no

[1] Site da ferramenta R: http://www.r-project.org

contexto em que a amostra é igual a 6 e o nível de confiança é de 99%, é de 0,3728, então a hipótese nula, ou seja, a hipótese que dita que os dados seguem uma distribuição normal, não foi rejeitada.

Tendo que os dados seguem uma normalidade, o conjunto de métodos indicado é o paramétrico, o que no caso deste trabalho foi escolhido o ANOVA, que foi apresentado na Seção 4.2.9.

O teste com ANOVA visa rejeitar a hipótese nula $H0_1$, definida na Seção 4.2.1, que afirma que não há diferença entre os métodos atual e proposto no que diz respeito ao TCS.

Executando o teste ANOVA através da ferramenta R, é possível extrair o *p-value* do resultado obtido:

p-value = 0,00895

Para um nível de significância de 99% é possível afirmar que a hipótese nula $H0_1$ foi rejeitada por cerca de 0,1%, o que é bastante significativo. O resultado permite afirmar que a hipótese $H1_1$ foi confirmada, tendo como fato que a média de TCS para o método atual é menor do que a do método atual, o que quer dizer que o método proposto mostrou-se mais produtivo que o método atual. Para corroborar o resultado estatístico e levantar algumas discussões positivas e negativas a respeito do método proposto, a seguir serão apresentados alguns dados qualitativos.

Dados qualitativos

Após o experimento realizado, foi aplicado o questionário do Apêndice D na Seção D.2, que avalia características como dificuldade e utilidade do método proposto através de uma pontuação de 1 à 10. No caso da utilidade 1 significa muito fácil contra muito difícil sendo 10, e no caso da utilidade 1 sendo muito pouco útil contra 10 sendo muito útil.

Os dados obtidos através do questionário foram:

- Utilidade: 9.25
- Dificuldade: 2.58

Analisando o resultado do questionário é possível avaliar que o método proposto apresenta bastante utilidade no entendimento da equipe de forma geral o que pode ser atestado estatisticamente no experimento analisado nas seções anteriores. Em relação à dificuldade, ficou claro que há um pouco de trabalho a ser feito em relação a dificuldade.

De forma geral, a pontuação da dificuldade não foi tão ruim mas alguns comentários reforçam que há muito o que melhorar.

Um participante citou o fato de haver uma interface que proporcionasse uma experiência mais fácil ou uma IDE que pudesse ter um pouco de *feedback* visual como *syntax highilighting*[2] e/ou uma funcionalidade de autocompletar. Com isso houve um tendência para o Selenium IDE por possuir uma interface. No entanto, também foi afirmado que com um pouco de conhecimento de programação, a curva de aprendizado da DSL do método proposto era pequena.

4.3 Conclusões

A fim de avaliar se a abordagem proposta é apropriada para o processo de teste da empresa, o gerente responsável pode avaliar os seguintes resultados obtidos neste estudo:

- O TCS avaliado no estudo mostrou que a equipe pode ser mais produtiva quando a variação de dados necessitar ser implementada num determinado teste ou conjunto de testes. O método apresentou resultados bem mais satisfatórias, havendo uma disparidade alta com alguns *testers*;
- A interação com a equipe através da análise qualitativa mostrou que de fato o método proposto merece atenção por sua utilidade e fácil utilização salvo algumas certas melhorias.

4.4 Resumo

Este capítulo apresentou um estudo experimental, que buscou avaliar o método proposto apresentado no capítulo anterior contra o método atual. Foi definida uma metodologia para a realização do experimento e ao fim foi possível observar que o método proposto possui uma certa vantagem em relação ao método atual, tendo como prova o teste estatístico realizado e a análise qualitativa com os participantes.

[2]http://en.wikipedia.org/wiki/Syntax_highlighting

5
Trabalhos Relacionados

Do you wish to rise? Begin by descending. You plan a tower that will pierce the clouds? Lay first the foundation of humility.

—ST. AUGUSTINE

Esta seção apresenta os Trabalhos Relacionados a esta dissertação, que serviram com fundamentação para a realização do mesmo, apontando aspectos muito relevantes e possíveis falhas a serem tratadas. Na sequência são apresentados estudos presentes na literatura na área de Testes Automatizados de Aplicações Web.

5.1 Automação de Testes de Aplicações Web

A indústria de *software* necessita constantemente evoluir, seja em gestão de recursos, projetos dentre outras particularidades. No âmbito de testes inserido nesta indústria não é diferente, é uma grande área do ramo da computação e também de grande importância, pois é nela que se verifica a qualidade do produto em relação as exigências de especificações, termos legais e de satisfação dos clientes envolvidos dentre outros aspectos. Então, partindo do que se propõe este trabalho, faz-se necessário um mapeamento do que há de pesquisas neste meio, com o propósito de buscar pontos de melhoria para o estado atual. Portanto, em seguida serão abordados estudos que atuam nesta área de pesquisa e ao final será apresentada uma categorização dos estudos.

5.1.1 Ferramentas de Captura/Repetição

Segundo Bertolini and Mota (2010), interfaces mudam de forma frequente no decorrer do desenvolvimento e produção. Este tipo de ação é comum por parte dos usuários que

sempre propõem qualquer sorte de alteração que caiba na sua proposta. É interessante que quando isso ocorra, entidades dependentes dos requisitos também sejam atualizados, o que garante a execução correta dos testes. Esta realidade é comum na indústria de *software* em geral.

Bertolini and Mota (2010) propõem uma abordagem que faz o mapeamento de uma linguagem de especificação de requisitos para *scripts* de teste, que tem como característica a técnica de captura/repetição, ou seja, a ferramenta é capaz de gravar eventos de interface que possam ser reproduzidas para testes de funcionalidade, regressão e aceitação. No entanto, um ponto negativo, que dificulta a validação da proposta por outra pesquisa está no fato de a ferramenta ser proprietária.

A linguagem CNL, discutida em Bertolini and Mota (2010), facilita a elaboração de casos de teste e requisitos. A ferramenta, porém, tem característica de linguagens naturais, o que onera em complexidade.

Uma das questões que este trabalho aborda e que é tratado em Bertolini and Mota (2010) é o tratamento de entradas para as requisições a serem feitas na interface. O estudo discute a possibilidade de anexar uma tabela de dados válidos e inválidos ao caso de teste para eliminar a ação manual neste estágio de execução do teste, o que este trabalho propões e que é apresentado no Capítulo 3. O estudo também comenta sobre uma possível geração automática de dados que pode ser tratada como um estudo futuro.

Bertolini and Mota (2010) diretamente apresenta as vantagens e desvantagens de utilização de seu produto. As principais vantagens consistem de: fácil criação de interfaces e geração automática de *scripts* de teste. Por fazer mapeamento de linguagens de especificação de requisitos e *scripts* de teste, o estudo também proporciona extensão para o suporte de variadas linguagens de *script*. A vantagem direta está no fato de suportar diferentes tipos de aplicações, seja *web*, *desktop*, *mobile*, etc. A desvantagem apresentada está na necessidade de se implementar o próprio *oracle*[1] ou o usuário pode utilizar um fornecido com a execução dos *scripts*. O estudo apresenta como trabalho futuro uma proposta interessante, que consiste em extrair requisitos do sistema e então gerar casos de teste e testar o sistema. Em Bertolini and Mota (2008) foi feita uma abordagem nesse sentido para aplicações *mobile*.

O trabalho em Paiva *et al.* (2005) segue a mesma linha de pesquisa e também faz uso de Spec# Microsoft (2012) para elicitar especificações.

Jin *et al.* (2008) relaciona algumas ferramentas de captura/repetição disponíveis na indústria, que denotam uma boa aceitação por parte dos praticantes da área. O trabalho

[1]Definição de *oracle* em testes de *software*: http://en.wikipedia.org/wiki/Oracle_(software_testing)

por si também segue o formato captura/repetição, tendo os dados gravados em XML, que então gera os casos de teste. O trabalho Li *et al.* (2008) também segue a mesma abordagem.

Uma abordagem diferente que pode ser caracterizada como captura/repetição está na pesquisa de Xu and Xu (2007), o qual toma como aliado Agentes Inteligentes para a captura de interação de usuários com o sistema a partir do *front-end* da aplicação. O benefício proposto neste estudo está na capacidade de aprendizado de comportamento de usuários na utilização do aplicativo por parte dos Agentes, que podem propiciar uma enorme variabilidade e aproximação de uma situação real a partir dos testes que estes podem automatizar.

Um exemplo de uso da ferramenta proposta em Xu and Xu (2007) seria incluir no *front-end* da aplicação a ser testada um Agente que grava as ações do usuário e manteria estes dados coletados como uma sequência de comportamento para utilização nos testes. Isto seria uma alternativa ao uso do *plugin* do Selenium no Firefox. No caso o usuário teria a sua ação gravada ao entrar no site. Em tempo de desenvolvimento poderia se implantar um módulo contendo os Agentes que proveria os testes automatizados seguindo especificações de testes, e em produção poderia se analisar comportamentos de usuários, o que ferramentas como Google Anlytics fazem e muitas agências de publicidade utilizam como técnica de aferição de comportamento dos potenciais clientes.

O trabalho em Qian *et al.* (2008) propõe uma ferramenta que faz uso do Watir[2] e de uma biblioteca para manipulação de planilhas do Excell. Dados são colocados na planilha que servem de entrada e também possuem dados que representam o resultado esperado para validação e uma espécie de graduação de acordo com a conformidade dos testes. A Figura 5.1 demonstra o funcionamento da ferramenta criada no trabalho.

Uma alternativa à criação tradicional de *scripts* é apresentada em Qian *et al.* (2008). Nota-se que não há criação de *scripts* de teste, ao invés disso, faz-se uso de planilhas para definição estrutural e de dados a serem utilizados de entrada e saídas esperadas. O maior ganho obtido com esta ferramenta proposta é o aumento na capacidade de estudantes aprenderem com erros auxiliados por ela, obtendo rápido *feedback*, permitindo assim um maior acompanhamento do seu progresso. O trabalho futuro proposto referencial Engine de Inferência Bayesiana para melhor diagnóstico a partir de experiências passadas.

Um outro *software* com características semelhantes ao Watir e ao Selenium é o Sahi[3], o qual pode ser visualizado na Figura 5.2. Certamente é uma alternativa bastante aceitável

[2]Site do Watir: http://watir.com
[3]Site do Sahi: http://sahi.co.in/w/sahi

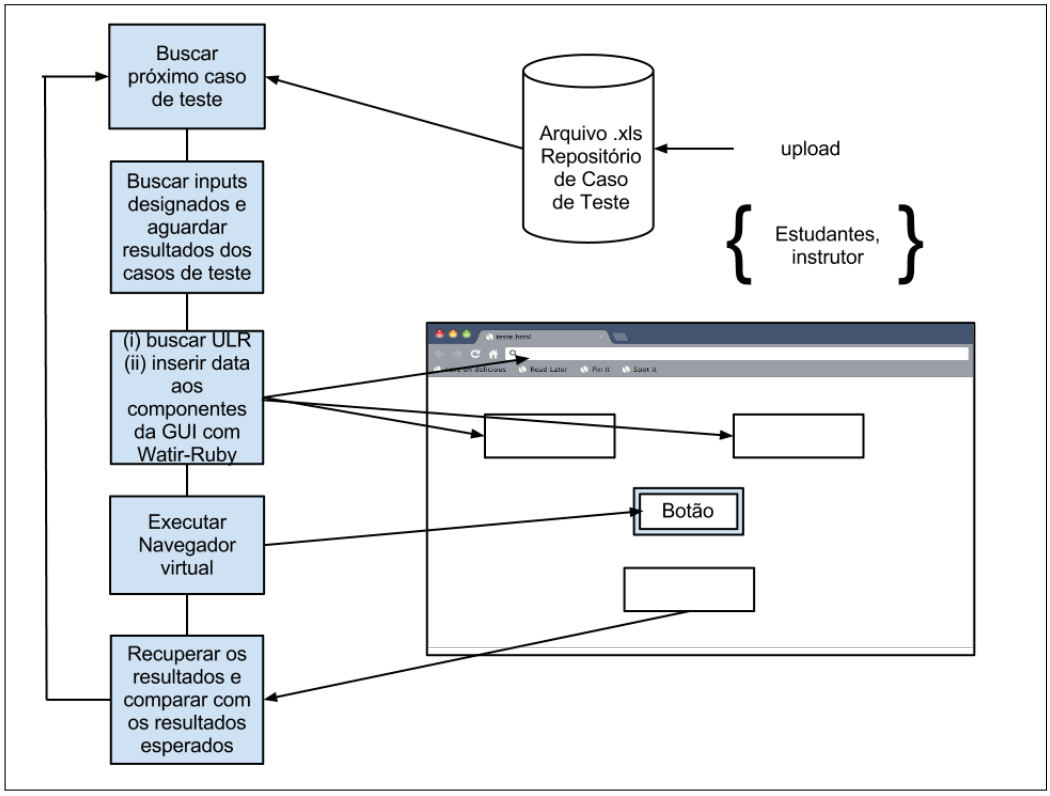

Figura 5.1 AWAT: ferramenta de automação de testes de aplicações *web*. Fonte: do autor, adaptado de Qian *et al.* (2008)

ao Selenium.

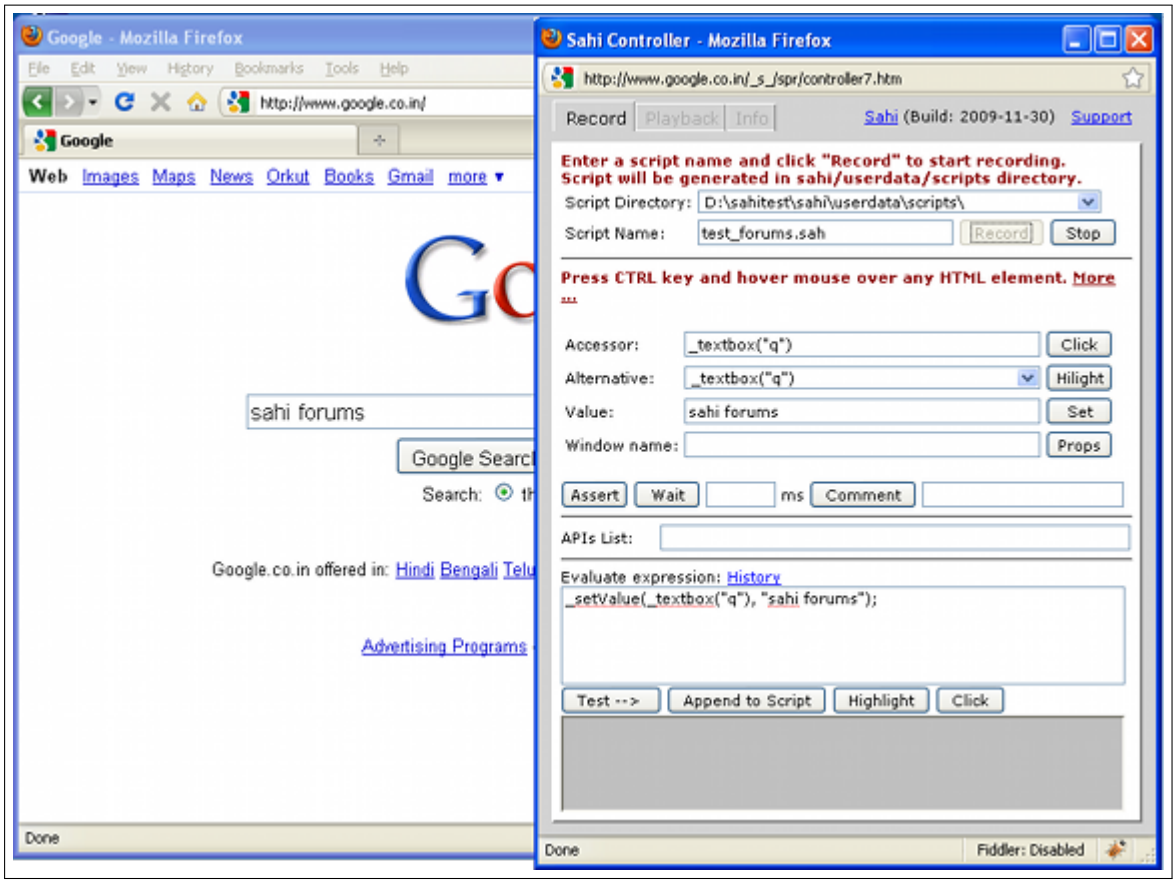

Figura 5.2 Sahi: ferramenta de automação de testes de aplicações *web*. Fonte: http://sahi.co.in/w/screen-shots

Nguyen *et al.* (2010) explica o surgimento da ferramenta Selenium, que apareceu como uma proposta de código aberto contra as ferramentas proprietárias como a WinRunner da Mercury Interactive e Robot da Rational. Os autores tentaram algumas ferramentas como Canoo WebTest e HTTPUnit mas não tiveram sucesso por falta de capacidade de manipulação de Javascript. A ferramenta proprietária QuickTest Professional da Mercury soluciona o problema do Javascript mas sofre de falhas se pequenas mudanças são efetuadas e não possibilita o Test-First. Houveram dificuldades de organização de documentação dos casos de teste, como escrita de testes em Tabelas.

Por ser de código aberto, Selenium possibilita a criação de extensões que podem viabilizar o seu trabalho de acordo com o desejado, como no caso exposto pelo trabalho: os testes tornaram-se mais legíveis. Indicação visual foi um atrativo para os clientes, e o fato de o Selenium conduzir testes de regressão, a equipe esteve mais confortável mesmo com rápidas alterações que a aplicação sofreu.

5.1.2 Alternativas às Ferramentas de Captura/Repetição

Modelos UML e Semelhantes

O trabalho Bouquet *et al.* (2008) expõe uma ferramenta capaz de gerar testes automatizados através de modelos de um subconjunto da UML com alvo de destino sendo várias plataformas. Faz uso de diagramas de classe para a determinação da estrutura dos componentes, como exemplo, a ferramenta é capaz de avaliar um *layout* de uma determinada aplicação validando contra a especificação do modelo. Diagramas de objetos são utilizados para identificação do estado da aplicação em um determinado momento, como estado inicial ou a persistência de dados após uma ação especifica do usuário. Utiliza diagramas de estado para o fluxo de execução além de utilizar OCL para expressão de comportamento que a aplicação permite.

A ferramenta, do trabalho Bouquet *et al.* (2008), não pôde ser encontrada, mas provavelmente mudou de nome para CertifyIt da Smartesting smartesting (2012). A solução é obtida através de capacitação com a empresa. No site smartesting (2012) não foi possível visualizar opções de *trial*, também não apresenta opções de compra direta. O estudo também afirma que sua ferramenta é capaz de gerar *scripts* para variadas plataformas inclusive a ferramenta da HP: HP Quick Test Professional HP (2012). A ferramenta automatiza teste para aplicações com interfaces ou não, apresentando o seu principal benefício, o rápido *design* de testes de uma forma intuitiva através de uma rica interface gráfica. No entanto, é inviável para esta pesquisa avaliar as qualidades da ferramenta, devido ao alto custo financeiro que deve ser despendido.

Em Huang and Chen (2006) também há uma bordagem voltada para modelos UML, baseando-se em extensões de modelo de atividade, fazendo uso de XMI para geração de teste para uma ferramenta chamada HttpUnit, que desde 2008 não é mantida, sendo que similares como HtmlUnit ainda estão presentes e é utilizado pelo Selenium. No entanto, estas ferramentas não simulam de forma realista a *engine* do Browser propriamente dito e sim uma outra *engine* mais simples e que carece de algumas funcionalidades presentes nos navegadores modernos. Outro problema da proposta, em comparação a este trabalho de dissertação é que geração de código de testes se destina ao Java, que necessita de um conhecimento razoável de programação e de bibliotecas da linguagem por parte de quem irá manipular, o que vai de encontro a proposta de facilitar o trabalho do engenheiro de teste com a automatização, a não ser que os envolvidos no projeto sejam qualificados para a função de desenvolvimento e teste, o que é comum em muitas empresas que não possuem recursos para custear especialistas.

A proposta desta dissertação, não envolve a necessidade de conhecimento de GPLs, ficando amarrada à uma plataforma específica para a automatização dos testes, mas sim o uso de arquivos de *scripts*, que serão manipulados por intermédio de uma linguagem de aparência familiar para os engenherios de teste que será implementada.

O estudo Huang and Chen (2006) comenta que com o uso de sua ferramenta é possível diminuir a carga de trabalhos triviais de programação na escrita de testes. No entanto ainda há a necessidade de envolver-se com uma linguagem de programação de propósito geral, o que não é o ideal.

O grande apreço à derivação de testes automatizados a partir de modelos UML é observado também em Turner *et al.* (2008). O trabalho apresenta um modelo de atividades que representam testes, sendo que as ligações indicam dependências entre casos de testes. A execução dos testes acontece da seguinte forma: através de uma entrada de URL a ferramenta pesquisa no HTML, *input tags* que devem ser preenchidos, neste caso aleatoriamente, envia para o servidor e checa os resultados. Os testes são gerados a partir da leitura dos *inputs* dada uma URL. O usuário pode modificar o código da forma que lhe for apropriado. Um exemplo seria uma página HTML que possui um formulário, o qual o usuário deve preencher. Pode-se automatizar este cenário da seguinte forma: buscar no HTML, *tags input*, preencher com dados aleatórios, enviar dado para o servidor e validar os resultados. As etapas de pesquisa no HTML e preenchimento automático são duas abordagens diferenciais que tem um alto potencial de automatização de trabalhos repetitivos, inclusive em ferramentas captura/repetição.

O problema que o estudo, discutido no parágrafo acima, incorre, está no fato da simulação de execução de navegadores por meio de ferramentas com HttpUnit, problema pertinente à algumas ferramentas que não fornece suporte adequado para algumas funcionalidades de Javascript presentes em navegadores modernos, o que prejudica a validade dos resultados.

Ricca and Tonella (2001) segue também pelo mesmo caminho, focando na criação de modelos UML para análise e utilização para casos de teste. A geração de casos de teste tem como origem expressões algébricas advindas de modelos Test-Flow com expressões de caminho. Faz uso de duas ferramentas, ReWeb que captura a aplicação e transforma em modelos UML. TestWeb consome os modelos para a realização dos testes. No entanto, é importante ressaltar que o processo não é 100% automático devido a necessidade de interação do usuário em algumas etapas do processamento.

Um dado interessante em Ricca and Tonella (2001) é que a ferramenta apresentada faz uso de *web spiders* (ou *web crawlers*) para obtenção de dados relacionados à aplicação

a ser testada. Houveram vários testes no estudo, em variados sites pela *web*, como, por exemplo, experiências com o site da Amazon. Contudo, o trabalho propõe, primordialmente, análises estáticas dos sites, ou seja, faz análise de componentes da interface e suas ligações, não atribuindo importância à análise funcional dos mesmos.

Métodos Formais

O estudo Zhu *et al.* (2009) fala sobre métodos formais para descrição de requisitos para a geração de casos de teste sem que haja problema de ambiguidade, por exemplo. O propósito deste artigo está muito relacionado a esta proposta de mestrado, que consiste em criar uma linguagem para descrever os casos de teste. A diferença é que o artigo propõe uma linguagem formal, não muito acessível, o que aumenta a curva de aprendizado, por envolver complexidades matemáticas. A proposta deste trabalho de dissertação propõe uma linguagem mais acessível para especialistas do domínio que não necessariamente dominem linguagens formais, o que de forma geral, não acontece.

Segundo Nguyen *et al.* (2010), a técnica de captura/repetição apresenta um problema relevante. Sendo este na manutenção, pois caso mude o *layout* pode haver perda, levando-se em consideração que a busca do componente é feita por coordenada, o que nem sempre é verdadeiro. Há varias opções que buscam por identificadores. A afirmação, no entanto, está incompleta o que é uma falha de avaliação. Destaca que modelos formais são uma solução para o problema.

Nguyen *et al.* (2010) também critica os trabalhos relacionados, indicando que estes possuem um laborioso processo de modelagem, além de serem dedicados ao teste da interface. Contudo, o trabalho apresenta esforço em dois modelos e uma linguagem para mapear, assim entrando em contradição à diminuição de esforço. A abordagem tenta solucionar dois problemas, primeiro resolvendo regras de negócio, chegando ao nível de código e utilizando estes teste para mapear ações na interface. A Figura 5.3 apresenta o diagrama de operação da solução.

Dificuldade para modelar uma aplicação pode ser substancial. Uma ferramenta como WordPad pode conter 121 eventos para serem modelados. Aplicações contendo interface gráfica possuem vários possíveis cenários, o que torna inviável testar tudo. O trabalho de Nguyen *et al.* (2010) propõe um critério de cobertura. O estudo incorre num custo extra no desenvolvimento do mapeamento de ações, mas há indicações no trabalho que indicam que o custo é menor em relação à abordagem tradicional.

O modelo de ações construído em Nguyen *et al.* (2010) foi feito transformando requisitos em texto para um modelo formal em Spec#. O modelo tem a forma de uma

Finite State Machine (FSM) e tem uma linguagem aproximada a uma GPL. A grande vantagem está na reutilização de casos de teste da lógica do negócio para a Graphical User Interface (GUI). O teste feito pode ser traduzido em eventos na GUI. O modelo da GUI utiliza Quick Test Pro, ferramenta proprietária da HP. Que utiliza captura/repetição para gerar o modelo baseado na utilização do usuário. Casos de teste são gerados para o Quick Test Pro.

Outro dado relevante do estudo Nguyen *et al.* (2010) consiste na análise de dados obtidos. A análise foi feita a partir do cálculo de esforço em horas de produção e tamanho dos arquivos (Lines of Code (LOC)). Defeito encontrados também foram comparados. A fraqueza do trabalho está no objeto estudado, que foi pequeno para se ter ideia real da vantagem de sua utilização. A geração de dados automática está como trabalho a ser desenvolvido, comum a alguns outros trabalhos relacionados, porém não foram encontrados estudos com esta abordagem de forma concreta.

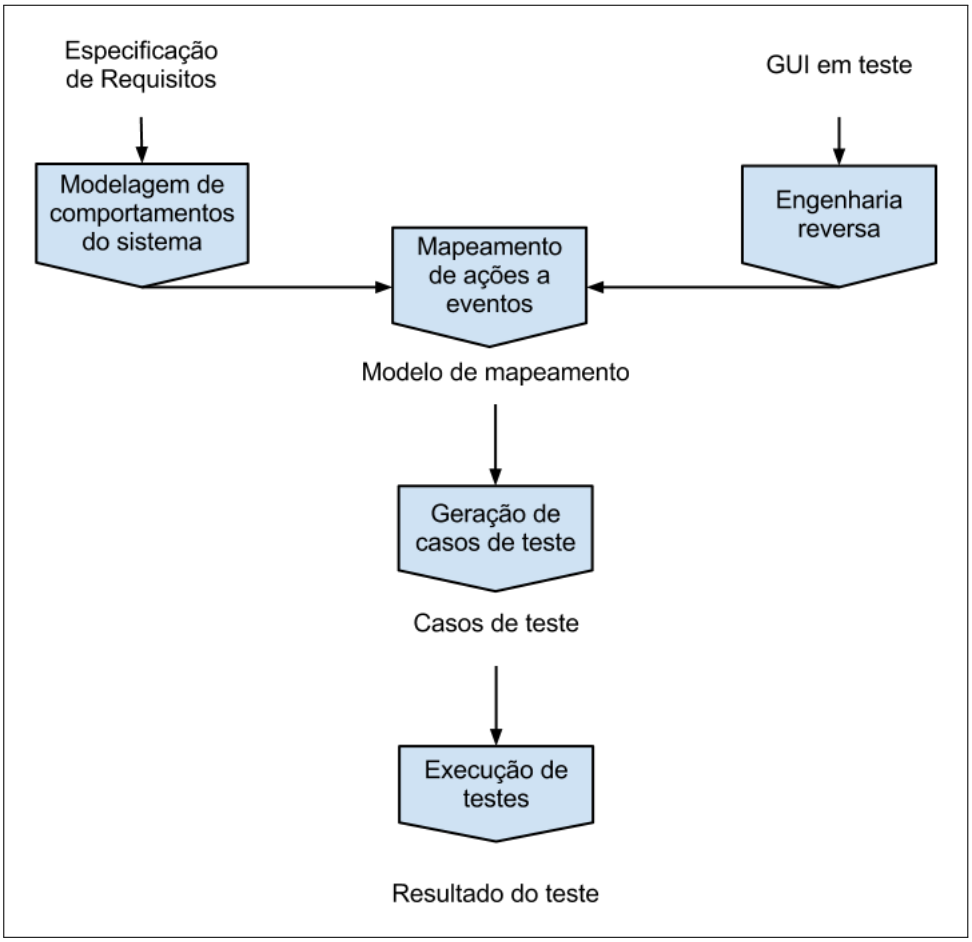

Figura 5.3 Action-Event Framework: diagrama de operação. Fonte: do autor, adaptado de Nguyen *et al.* (2010)

O trabalho Barbosa *et al.* (2011) foca nos erros de usuários e examina a viabilidade no uso de modelos de atividade para testar seguindo esse comportamento dos usuários. Baseia-se em Silva *et al.* (2008) e a abordagem usa ConcurTaskTrees, presente em Paternò (1999), como notação.

Barbosa *et al.* (2011) também destaca os problemas de testar manualmente, custos de mão-de-obra e suscetibilidade a erros. O estudo também enfatiza que métodos de teste baseados em modelos apresentam muitas dificuldades, dentre elas, relativa a sistemas interativos, a necessidade de se elaborar modelos detalhados. Uma forma de resolver é aumentar a abstração do modelo. Em Silva *et al.* (2008) é apresentado um método baseado em modelos de tarefa que supera este problema, porém, este método descreve somente operações normativas, não capturando erros de usuários ou caminhos alternativos. Resolvem o problema do custo de produção agregando as ferramentas em um único produto.

Ainda, o trabalho Barbosa *et al.* (2011) incorre no problema do conjunto ferramental extenso para a execução do processo automatizado de teste. Modelagem, exportação para XML com uma ferramenta, introdução de erros a partir da ferramenta CMTTool desenvolvida no trabalho, novos XMLs gerados com as mutações respectivas FSMs, novamente utilizando Teresa para transformação para XML. Passa por Spec#, Tom e finalmente por Spec Explorer para gerar os casos de teste. Em suma, a ferramenta tem como entrada modelos de aplicações GUI, que passam por uma mutação através de um algoritmo que utiliza um classificador de erros de usuário chamado Reason. Novos modelos são gerados com comportamentos baseados no classificador.

Alguns pontos finais devem ser levados em consideração em relação à validade do trabalho Barbosa *et al.* (2011). Os cenários utilizados na ferramenta foram de pequena proporção, portanto não há como ter conhecimento sobre escalabilidade, além da custosa tarefa que envolve várias ferramentas no processo de geração e execução de teste. O Mapeamento da GUI, em particular utiliza bibliotecas proprietárias o que dificulta a reprodução do processo. Outro aspecto é que os autores não tiveram acesso a dados reais para comparação de erros de usuário que a ferramenta utiliza através de uma classificação. Como trabalho futuro a pesquisa discute a possibilidade de automatização a geração de entrada de dados através de modelos formais.

Outras Alternativas

Bruns *et al.* (2009) apresenta um trabalho relacionado a esta dissertação no sentido de fazer uso de técnica similar, porém a sua abordagem apresenta certa deficiência já

discutida anteriormente. O estudo discute que ferramentas como HTTPUnit estão ao nível de protocolo, ou seja, é uma requisição HTTP para uma determinada URL, e como resposta espera-se geralmente uma página HTML. No entanto, interfaces gráficas modernas são altamente dinâmicas e fazem uso massivo de AJAX, característica marcante na Web 2.0, e a abordagem de ferramentas como HTTPUnit não suportam tal dinamismo, por deficiências de suporte a Javascript, por exemplo. Selenium dá um salto do nível de protocolo ao nível de interação de usuário com a aplicação. Selenium faz uso das capacidades do navegador em que ele atua. O artigo menciona WebDriver do Google, que no momento está integrado ao Selenium, chamando-se Selenium 2 WebDriver. Apresenta duas ferramentas, CubicTest e Testmaker, as quais tem como base o Selenium e serão discutidas em outra seção.

O estudo Pava *et al.* (2009) propõe uma ferramenta para identificação de plataformas de desenvolvimento *web* e então gerencia os casos de teste para esta plataforma. O cenário consiste em ter uma aplicação em uma plataforma X, ter os testes para esta plataforma e em algum momento haver uma alteração da plataforma, para Y. A ferramenta identifica a nova plataforma e automatiza a geração de *scripts* de teste para a nova plataforma dado um modelo independente de plataforma previamente concebido. No entanto, a proposta abrange uma área distinta deste trabalho, não deixando, contudo, de se exaltar o intuito da pesquisa que apresentou-se como uma excelente proposta para se combater a questão da manutenção dos testes existentes, quando há a troca para uma outra tecnologia, mais especificamente uma linguagem distinta. Com a transformação dos testes, é possível ter o reuso e a economia do retrabalho e consequentemente de custos financeiros e de mão-de-obra.

Sampath *et al.* (2011) foca na transformação de *logs* de uso do sistema por usuários para a geração de casos de teste, estes em formato XML, que tem como característica ser muito verbosa, que poderia ser amenizada por uma DSL. A abordagem do estudo se mostra muito interessante, como uma alternativa às ferramentas captura/repetição, pois torna o processo indireto e não necessita de uma ferramenta de auxílio à gravação. Contudo, os testes da aplicação estão a nível de protocolo, ou seja, os teste são avaliados a nível de HTTP, com o acréscimo da análise de requisições HTTP POST, através de implementação de um módulo para o servidor de aplicação *web*. Relaciona alguma outras ferramentas como *wget*[4] do Linux, e sobre protocolos, requisições, etc.

O trabalho de Sampath *et al.* (2011) também dá importância à priorização dos casos de teste, devido ao alto custo de se executar toda uma suíte de testes, acumulados ao longo

[4]Informações sobre o *wget*: http://www.gnu.org/software/wget/

de um longo período. O estudo também discute que há uma escassez de ferramentas facilmente disponíveis que possuem técnicas de priorização, sendo que a pesquisa propõe uma solução nesse sentido. Utilizam heurísticas proveniente de Elbaum *et al.* (2003) para geração de casos de teste. Para controlar uma grande quantidade textitlogs, que há em aplicações web, os autores optaram por utilizar banco de dados.

Em outra vertente, o trabalho de Somé and Cheng (2008) busca derivar casos de teste a partir de casos de uso. Este estudo possui uma implementação não concluída, portanto, um protótipo, o que não caracteriza uma desqualificação pois sua proposta descende de uma ideia bem promissora. Todavia, existem alguns problemas ao tratar casos de uso para a geração de casos de teste. Primeiro que os casos de uso são escritos em linguagem natural informal, o que torna o tratamento muito complexo. Um segundo desafio está em se identificar a cobertura que o caso de uso proporciona, e um outro problema é que restrições ou regras de sequência entre casos de uso geralmente são tratadas implicitamente, por conjecturas de quem os elabora.

Para solucionar o problema da natureza informal dos casos de teste, Somé and Cheng (2008) referenciou o trabalho Somé (2006) que descreve uma linguagem restrita. Utilizam Máquinas de Estados Finitas (FSM) para controle de cobertura das sequências dos casos de uso. Fazem inferências de relações de casos de uso comparando precondições e pós condições. Isso permite a combinação de comportamentos. A linguagem proposta na pesquisa tem caráter textual, assim como a proposta neste trabalho de dissertação. Um exemplo para o uso de FSM e DSLs no contexto de linguagens pode ser encontrado em Fowler (2011), que serve de referência para a DSL criada nesta dissertação.

Uma crítica relevante observada no estudo acima discutido e o que é notável em muitos outros trabalhos, é que muitos estudos atacam o problema utilizando um ferramental complexo que dificulta o entendimento de tantos artefatos e posteriormente a manipulação laboriosa e consequentemente custosa em tempo e mão-de-obra. Uma alternativa plausível é centralizar ao máximo como meio de organização, e manter a curva de aprendizado o menor possível para alcançar com maior agilidade o objetivo.

Um outro ponto muito interessante consiste numa exacerbação de propósito, ocasionada muitas vezes por impulso, de automatizar completamente um processo de qualquer natureza, o que prova ser impraticável. Certamente, há muitas variáveis envolvidas num sistema, que o tempo a ser despendido na tentativa de se automatizar de forma completa não compensa a sua criação.

O trabalho Schwarz *et al.* (2005) apresenta mais uma variação de geração de casos de teste através de modelos, não especificamente UML. Neste caso um ganho está na

inclusão de benefícios de edição integrado à geração de *scripts* de teste permitida pela IDE Eclipse contendo *wizards* e editores gráficos próprios. A persistência do arquivo é em arquivo XML que permite controle de versão adequado. Sua *engine* contém uma ferramenta similar ao HtmlUnit, no entanto, aparenta ser inferior, contendo problemas maiores de Javascript e HTML, a ferramenta chama-se JWebUnit WebFixture, que é uma extensão do FIT. A simulação do Selenium, Watir e Sahi podem propiciar resultados mais realistas por utilizar as *engines* dos próprios *browsers*.

Comumente sistemas são desenvolvidos sem a prática do teste da forma que deveria ser feita. Perde-se uma grande informação para desenvolver o que se pretende, pois um caso de teste conta uma história de como é a funcionalidade. Outro problema é a falta de sincronia entre requisitos e os testes. Ainda, há pouca priorização para uma prática tão importante.

O estudo Li *et al.* (2011) utiliza agentes para uma série de tarefas dividas em 2 módulos. O primeiro é a a parte de modelos, no qual agentes com tarefas específicas desempenham tarefas com intuito de obter dados da aplicação: códigos, especificações, etc. Estes modelos servem de entrada para o módulo de teste que é responsável por gerá-los em *templates* e executá-los. Um detalhe importante do estudo é que o *framework* apresentado é apenas um esboço, não há, portanto, uma aplicação real.

A ferramenta FLOAppTest, apresentada no estudo Araújo *et al.* (2007), alega que testa não somente aplicações Web ou GUI, mas qualquer aplicação Java. Em se tratando de Java essa afirmação não parece tão favorável contra às outras. Sua ferramenta é baseada em EasyAccept Sauvé *et al.* (2006). Utiliza característica do FIT: tabelas de resultados esperados, incluindo outras. Apresenta vantagens de se fazer teste de aceitação e discute o fato de que clientes estejam envolvidos em escrita de documentos em linguagem natural. Destaca também o envolvimento desses *stakeholders* na participação da escrita dos testes executáveis além de simples comunicação, preza a interação ativa. O que abre um forte espaço para a inclusão de uma linguagem acessível mas ainda assim restritiva, eliminando a complexidade extra contida em linguagens naturais.

O estudo Araújo *et al.* (2007) também propõe criar uma ferramenta de execução de teste em um ambiente colaborativo via Web Services, eliminando recursos excedentes de configuração, tornando-o acessível de qualquer localização que tenha acesso a internet. Por ser uma aplicação Web, a ferramenta não necessita de instalação de um ambiente, a não ser é claro no servidor da aplicação. Utiliza Reflection API para leitura de testes disponíveis segundo a classe Façade, o que torna a aplicação dependente de uma classe seguindo este padrão para prover interação.

O que pode ser notado no trabalho de Im *et al.* (2008) é o uso de uma cadeia de ferramentas para a realização do trabalho, dividida em uma para a elaboração dos casos de teste, outra para o *feature model*, outra para design de *templates* e assim sucessivamente. Uma dificuldade, que se não existisse haveria um ganho na avaliação, foi a não apresentação da linguagem criada por nenhum aspecto, apenas menciona-se uma ferramenta para extração de nomes e verbos que são utilizados para a criação de programas na DSL. Utiliza-se a ferramenta Abbot [5] que exclusivamente é para aplicativos que tem como plataforma o Java, especificamente para *desktop*.

A proposta de Im *et al.* (2008) assemelha-se a este trabalho de dissertação no sentido de discutir a utilização de DSLs e possui argumentos muito fortes para o uso de SPLs (Linhas de Produto de Software) , como a justificativa para o uso de DSL para uma única aplicação não ser apropriada mas sim para uma família de produtos. Primordialmente, o ganho está na especificidade ao focar em uma linha de produto. O custo maior está na etapa de construção da linha de produto, posteriormente eliminando esforço nos produtos gerados. O ponto negativo encontrado foi a série de ferramentas que foi utilizada para a geração dos casos de teste, mas de certa forma é justificável, por se tratar de reuso, porém há um custo maior de aprendizado. A outra parte negativa é o alvo de aplicações restritas como o Java para Desktop em prol de uma maior cobertura como a plataforma Web, a qual tenho como alvo neste trabalho de dissertação.

O estudo Ran *et al.* (2009) está relacionado a este trabalho de dissertação à primeira vista, no entanto, a pesquisa apresentada busca conformidade com dados presente em bancos de dados, ou estado de persistência, ao invés de somente checar o fluxo de interação do usuário de forma genérica, angariando uma melhor verificação de consistência de dados na perspectiva do *front-end*. O trabalho esbarra na necessidade de se modelar uma Máquina de Estado, de acordo com os requisitos e o estado do banco de dados da aplicação. Utiliza a linguagem *prolog* para definição de restrições de entrada e estado e atualizações do banco de dados e atenta para o fato de alguma aplicações web terem dependência de alguns dados que podem estar disponível ou não, o que torna o teste complicado.

Com outra abordagem, Guillemot and König (2006) apresenta uma ferramente de automação de testes. Um dos pontos discutidos que denotam um benefício em sua utilização é fácil escrita dos testes por se tratar de XML. No entanto, XML é ainda muito verbosa em relação a uma DSL mais aproximada ao idioma do meio. Muitas vezes, XML pode se tornar um vilão a medida que o conteúdo cresce em tamanho. Menciona

[5] Site da ferramenta Abbot: http://abbot.sourceforge.net/doc/overview.shtml

ainda teste *end-to-end* e introduz sua ferramenta WebTest, de código aberto, a qual foca em *features* relacionada a produtividade. A ferramenta tem um alto alcance para testes de diversas tecnologias, além do próprio HTML como AJAX, *applets* Web Services, PDF e planilhas Excel. É combinada ao Ant, que é bastante utilizado na indústria para automação de construção, implantação e teste. WebTest habilita extensão, o que favorece a especificidade de práticas aplicadas em situações diferenciadas. A ferramenta possui um módulo de geração de relatórios que tem como arquivo de destino um XML que pode ser transformado em um HTML contendo dados detalhados da execução.

É interessante mencionar que Guillemot and König (2006) recomenda algumas boas práticas como o uso de IDs nos elementos a serem testados para facilitar a formulação dos testes, sendo que é necessária uma colaboração das equipes de desenvolvimento e de teste. Essa colaboração poderia ser através de UI Mapping. Preza organização dos testes, como nome e parametrização. Uma vantagem está na abordagem declarativa que torna a leitura dos teste melhor, porém ela poderia ser ainda melhor se tivesse uma expressividade mais restrita para testes, ou seja, apresentasse construções unicamente voltadas para este domínio.

O problema da ferramenta vista em Guillemot and König (2006) é ter como núcleo de execução o HtmlUnit quando se trata de simulação, que é sabido que não contém funcionalidades de Javascript que algumas *engines* possuem, problema este comum à alguns trabalhos já citados. Porém, o autor argumenta que o suporte a HTML e validação deste de forma contunde é um enorme ganho pois como auto discute, erros de HTML ocasionam erros em alguns *browsers* e causa problemas para aplicações de acessibilidade e leitores de tela, o que pode acarretar problemas para aplicações muito dinâmicas. Crítica ferramentas de captura/repetição como Selenium IDE, que são ruins em manutenção por haver necessidade de se gravar os movimentos novamente. Criar um módulo auxiliar para escrita das alterações é uma solução, como uma linguagem mais simples que o WebTest. Seria obtido então uma *feature* capture/replay aliada a uma DSL que possibilitasse de maneira simples a alteração dos *scripts*.

Uma outra ferramenta, a WebAppML, da Conformiq[6], que tem como base ferramentas como Selenium e JUnit para execução dos testes e Metacase para a criação da DSL que viabiliza a elaboração dos testes e que permite a criação de *scripts* e documentação. Uma demonstração da ferramenta pode ser visualizada na Figura 5.4.

[6]Site da Conformiq: http://www.conformiq.com

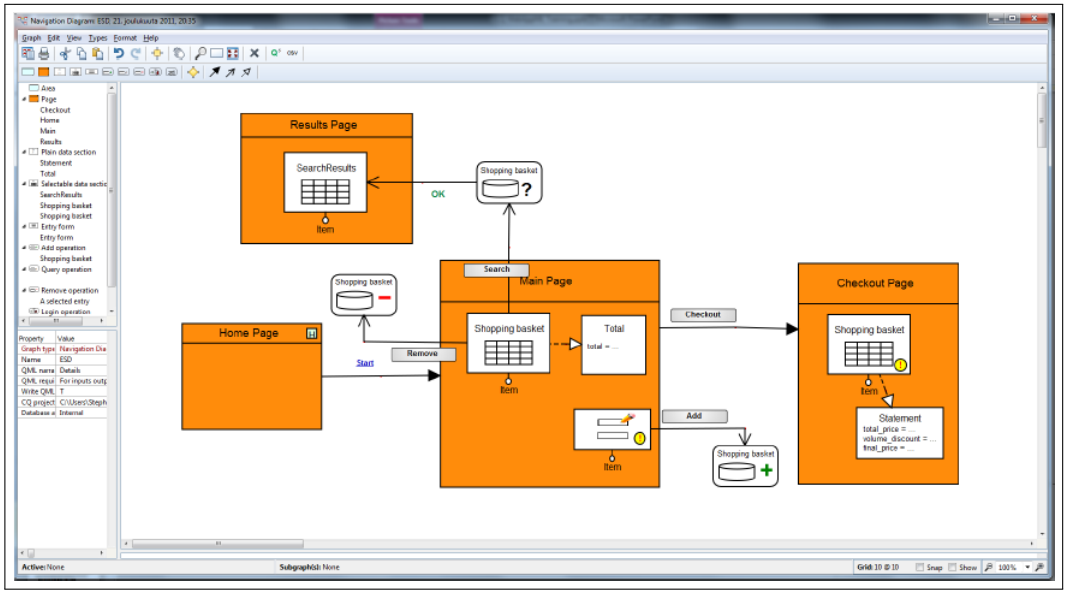

Figura 5.4 WebAppML: ferramenta para design e execução de testes de aplicações *web*. Fonte: http://www.metacase.com/blogs/jpt/blogView?entry=3517490549

5.1.3 Categorização dos Estudos de Testes Automatizados

Nesta subseção poderá ser observado, de forma mais evidente, o agrupamento de pesquisas seguindo linhas relacionadas com o intuito de reunir as informações para uma análise mais facilitada dos assuntos, com seus pontos fortes e outros negativos que levam a identificar com maior clareza as lacunas pertinentes à área de estudo. Portanto, na Tabela 5.1, será apresentada uma categorização dos trabalhos apresentados referentes aos assuntos discutidos relativos à testes automatizados de aplicações *web*. A primeira linha, em destaque, consiste na ferramenta apresentada nesta dissertação.

Tabela 5.1: Classificação dos Estudos

Estudo	Ferramenta/Técnica	Design	Plataforma	Licença
Este trabalho	Captura/Repetição	DSL	Web	Gratuita

Bertolini and Mota (2010)	Captura/Repetição	Linguagem natural	Web, Desktop, habilita extensão para diversos tipos de scripts	Proprietária
Paiva *et al.* (2005)	Spec#	Linguagens Formais		
Jin *et al.* (2008)	Captura/Repetição	XML		
Li *et al.* (2008)	Captura/Repetição	XML		
Xu and Xu (2007)	Captura/Repetição	Agentes Inteligentes no front-end	Web	
Qian *et al.* (2008)	Captura/Repetição (Watir)	Scripting em Ruby	Web	Código aberto
Nguyen *et al.* (2010)	Captura/Repetição (Selenium)	Scripting em Ruby, Java, C# e PHP	Web	Código aberto
Bruns *et al.* (2009)	HttpUnit	Scripting em Java à nivel de protocolo	Web	Código aberto
Bouquet *et al.* (2008)	UML e OCL (CertifyIt e HP Quick Test Professional)	Modelagem	Web	Proprietária

Huang and Chen (2006)	UML/XMI e HttpUnit	Modelagem	Web	Gratuita
Turner *et al.* (2008)	Modelo de Atividades e HttpUnit	Modelagem	Web	
Ricca and Tonella (2001)	UML (Test-Flow), ReWeb e TestWeb	Modelagem	Web	
Zhu *et al.* (2009)		Linguagens Formais		
Nguyen *et al.* (2010)		Linguagens Formais		
Barbosa *et al.* (2011)	FSM, Spec#	Linguagens Formais, Modelagem, XML		
Pava *et al.* (2009)		Transformação de scripts entre plataformas	Web e permite geração de scripts para várias plataformas diferentes	
Sampath *et al.* (2011)		Transformação de logs de utilização para XML	Web	

Somé and Cheng (2008)	Casos de Uso	Linguagem natural e FSM		
Schwarz *et al.* (2005)	Plugin do Eclipse (HtmlUnit e extensão para FIT)	Modelagem e XML	Web	
Li *et al.* (2011)		Modelagem		
Araújo *et al.* (2007)	Baseada em EasyAccept com características do FIT	Tabulação de dados de verificação e Linguagem natural	Qualquer plataforma suportada por Java	
Im *et al.* (2008)		SPL e DSL	Desktop (Java)	
Ran *et al.* (2009)		FSM, prolog e Banco de Dados	Web	
Guillemot and König (2006)	WebTest (HtmlUnit)	XML e HTML (relatórios)	Web	Código aberto

5.2 Resumo

O capítulo apresentou um conjunto de trabalhos relacionados a esta dissertação. Os trabalhos estão distribuídos por categorias de técnicas utilizadas para abordar os problemas vistos neste trabalho.

6
Conclusões

Teste de *software* é uma disciplina de suma importância para a indústria. É a partir de testes que se atesta um nível de qualidade pretendido, de acordo com o objetivo proposto no sistema criado. Portanto, muito tem-se falado e muito tem-se executado no sentido de promover uma maior qualidade de produtos e serviços no que diz respeito a *software*. A realidade é tangível, acessível globalmente, como metodologias e práticas que tornaram o tema teste alvo de grande foco, havendo muitos nomes que ressoam a disciplina, como Test-Driven Development (TDD) e metodologias ágeis que levam muito a sério o fator experimentação como Scrum, XP e Lean.

Ao nível operacional, muitas ferramentas que dão suporte a metodologias e práticas que norteiam o tema teste, tem sido criados para solucionar um problema muito recorrente em qualquer empreendimento que contenha *software*, seja na indústria ou na academia que muito consomem estes produtos mas muito produzem também, sejam soluções para uso de forma gratuita, seja de código aberto ou mesmo proprietários.

É inegável que existem muitas soluções proprietárias que resolvem problemas práticos de diversos tipos de problemas para consumidores em geral, porém é verdade também que há um leque diverso de soluções gratuitas que fornecem a solução em igual qualidade ou até melhor.

Um nicho especial que tem grande parcela de produtos e serviços que necessita de grande ajuda de soluções de testes para validar a qualidade desses é a *web*, que conta com uma infinidade de variações de aplicações com arquiteturas, com configurações de servidores, linguagens de programação de *backend* até tecnologias que permitem alta interatividade nas interfaces de usuário através de AJAX e diversos *frameworks* que habilitam tudo isso formando um turbilhão de tecnologias e informação adjacente, transformando todo o processo de desenvolvimento de *software* muito complexo, exigindo um alto investimento no esforço de se tratar tudo isso de forma que o produto ou serviço

final seja condizente com a expectativa e desejos do usuário final.

Para que *softwares* voltados para internet como um todo, também são necessárias ferramentas que possam permitir que esses sejam testados de forma eficiente. Geralmente estas mesmas ferramentas são fabricadas em cima da própria *web*, contendo ferramentas de testes de integração, de relatório de *bugs*, controle de casos de teste em ferramentas de gerenciamento entre muitas outras.

Esta dissertação teve como alvo os testes de regressão em uma empresa, a qual também tem objetivos e expectativas quanto a qualidade a ser transmitida para os seus consumidores. Para isso foi feita uma avaliação do método praticado na empresa e assim foi possível observar os detalhes que apresentavam alguns pontos negativos em termos de produtividade, que poderiam ser mitigados e em sequência propor uma solução para o problema. Por fim, tendo-se a proposta com um novo método foi feito um estudo e o resultado final é que houve muito ganho de experiência, por avaliar o estado atual e também houve um resultado satisfatório em termos de produtividade, resultado este obtido através de um experimento realizado internamente na empresa.

Em suma, as contribuições desejadas, citadas no Capítulo 1, na Seção 1.3 foram atingidas:

- Um novo método de teste de regressão foi criado baseado em uma Domain-Specific Language (DSL) para atingir o propósito de aumentar a produtividade, aliviando processos laboriosos que tomam muito tempo dos engenheiros de teste;
- O novo método permitiu também novas construções que permitem que testes possam ser executados de forma automática com muitas interações com tratamento de falhas e um simples sistema de *log*;
- Um estudo experimental que propiciou avaliar o método proposto em comparação ao método atualmente praticado na empresa com intuito de provar sua eficácia e maior apelo em relação a este, de forma que este possa ser substituído sem muitas implicações no processo de teste. O estudo contou com um método estatístico para atestar as afirmações e fundamentar o resultado, além de uma análise qualitativa obtida através de uma interação com a equipe de teste;
- Além disso, foi possível observar na Seção 2.2 um ganho de cobertura de testes em relação a plataformas, ou seja, com a solução proposta, mesmo que aparentemente simples, houve um acréscimo no alcance de teste, permitindo avaliar de forma mais completa a aplicação alvo.

6.1 Trabalhos Futuros

Como trabalhos futuros o estudo leva em consideração alguns pontos observados pelo autor e outros relatados pela equipe de testes da empresa onde o experimento foi aplicado:

- A linguagem apresenta uma qualidade satisfatória para a resolução do problema, porém há espaço para melhorar a construção de sua principal funcionalidade que é a utilização de fontes de dados para variação de entrada, ou seja, há como possibilitar uma melhor leitura da definição da variabilidade de entrada;
- Ainda em relação a Domain-Specific Language (DSL), foi sugerido que houvesse algum tipo de suporte de verificação de sintaxe como proporcionadas pela IDE Eclipse, o que sugere um provável *plugin* para ferramenta na IDE;
- É discutível também a ausência de elementos gráficos, o que pode haver no *plugin* supracitado, elementos gráficos que deem algum suporte a linguagem, tornando-a um híbrido de Domain-Specific Language (DSL) textual e Domain-Specific Modeling (DSM);
- Estender a ferramenta para automatizar a análise de *logs* de execução, o qual é feito ainda manualmente;
- Após as melhorias na linguagem é importante realizar um novo experimento para uma população maior e então avaliar o tempo de execução que deverá ter mais impacto numa amostra maior. Esta abordagem pode, então, permitir uma maior generalização do método, possibilitando que outras empresas na indústria e a academia possam tirar proveito.

6.2 Considerações Finais

Mesmo tendo conhecimento das ameaças a validade do trabalho, como a validade externa, que consiste em se generalizar o trabalho, este mostrou-se muito valioso, porque foi possível avaliar um problema prático de uma empresa da indústria de forma científica, avaliando-se dados estatísticos que fortalecem o resultado e assim possibilitou observar uma melhoria que pode ser mais viável no que diz respeito a produtividade da equipe de testes, tornando o processo mais prático. No entanto, como visto anteriormente, nos Trabalhos Futuros, há um esforço para que se possa evoluir com o método e consolidá-lo

para se tornar um elemento essencial no processo de testes da empresa e até mesmo maximizar o seu potencial de generalização.

Referências Bibliográficas

Araújo, B., Rocha, A., Xavier, A., Muniz, A., and Garcia, F. (2007). Web-based tool for automatic acceptance test execution and scripting for programmers and customers. In *Proceedings of the 2007 Euro American conference on Telematics and information systems*, page 56. ACM.

Barbosa, A., Paiva, A., and Campos, J. (2011). Test case generation from mutated task models. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 175–184. ACM.

Beck, K. (2002). *Test-Driven Development By Example*. Addison Wesley.

Beighley, L. (2007). *Head First SQL*. O'Reilly Media.

Beizer, B. (1995). *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. Wiley.

Bertolini, C. and Mota, A. (2008). Using Refinement Checking as System Testing. In *11th Iberoamerican Workshop on Requirements Engineering and Software Environments (IDEAS 2008)*, volume 1, pages 17–30.

Bertolini, C. and Mota, A. (2010). A Framework for GUI Testing Based on Use Case Design. *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, pages 252–259.

Bouquet, F., Grandpierre, C., Legeard, B., and Peureux, F. (2008). A test generation solution to automate software testing. In *Proceedings of the 3rd international workshop on Automation of software test*, pages 45–48. ACM.

Bruns, A., Kornstadt, A., and Wichmann, D. (2009). Web Application Tests with Selenium. *Software, IEEE*, **26**(5), 88–91.

Coleman, D., Ash, D., Lowther, B., and Paul Oman (1994). Using Metrics to Evaluate Software Svstem Maintainabilitv. *IEEE Computer*, pages 44–49.

Deursen, A. (1998). Little languages: Little maintenance? *Journal of Software Maintenance:*, pages 1–19.

Deursen, A. V. and Klint, P. (2000). Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, **35**(June), 26–36.

Dmitriev, S. (2004). Language oriented programming: The next programming paradigm. *JetBrains onBoard*, **1**(2).

Elbaum, S., Karre, S., and Rothermel, G. (2003). Improving web application testing with user session data. In *Proceedings of the 25th International Conference on Software Engineering*, pages 49–59. IEEE Computer Society.

Fowler, M. (2005). Language workbenches: The killer-app for domain specific languages. *Accessed online from: http://www. martinfowler. com/articles/languageWorkbench. html*.

Fowler, M. (2011). *Domai-Specific Languages*. Addison Wesley.

Fowler, M. (2012). Introducing domain specific languages. http://msdn.microsoft.com/en-us/oslo/dd727707.aspx, acessado em 30/01/2012.

Fowler, M. and Sells, C. (2012). Expert to expert: Martin fowler and chris sells - perspectives on domain specific languages. http://channel9.msdn.com/posts/Charles/Expert-to-Expert-Martin-Fowler-and- Chris-Sells-Perspectives-on-Domain-Specific-Languages/, acessado em 30/01/2012.

Guillemot, M. and König, D. (2006). Web testing made easy. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 692–693. ACM.

HP (2012). Hp quick test pro. http://www8.hp.com/uk/en/software/software-product.html?compURI=tcm:183-936981, acessado em 16/01/2012.

Huang, C.-h. and Chen, H. Y. (2006). A Tool to Support Automated Testing for Web Application Scenario. *2006 IEEE International Conference on Systems, Man and Cybernetics*, pages 2179–2184.

Im, K., Im, T., and McGregor, J. (2008). Automating test case definition using a domain specific language. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*, pages 180–185. ACM.

Jin, X., Xu, J., Jia, L., Tian, H., and Pang, B. (2008). A Web-App Auto-Testing System Based on Test-Flow and Control Constraints. *2008 International Conference on Computer Science and Software Engineering*, pages 702–707.

Juristo, N. and Moreno, A. M. (2001). *Basics of software engineering experimentation*.

Kam, B. and Dean, T. R. (2009). Lessons Learned from a Survey of Web Applications Testing. *2009 Sixth International Conference on Information Technology: New Generations*, pages 125–130.

Li, J., Jing, X., and He, T. (2008). An Automatic Execution System for Web Functional Test Base on Modelling User's Behaviour. *2008 International Symposium on Information Science and Engineering*, pages 263–267.

Li, J., Sun, L., Liu, S., and Wei, R. (2011). Web Applications Testing Framework based on Multi-Agent. *International Journal of Advancements in Computing Technology*, **3**(10), 403–412.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*, **37**(4), 316–344.

Microsoft (2012). Spec#. http://research.microsoft.com/en-us/projects/specsharp/, acessado em 16/01/2012.

Nguyen, D., Strooper, P., and Süß, J. (2010). Automated functionality testing through GUIs. In *Proceedings of the Thirty-Third Australasian Conferenc on Computer Science-Volume 102*, number Acsc, pages 153–162. Australian Computer Society, Inc.

Paiva, A., Faria, J., Tillmann, N., and Vidal, R. (2005). A model-to-implementation mapping tool for automated model-based GUI testing. *Formal Methods and Software Engineering*, pages 450–464.

Paternò, F. (1999). Model-Based Design and Evaluation of Interactive Applications. UK: Springer-Verlag.

Pava, J., Enoex, C., and Hernandez, Y. (2009). A self-configuring test harness for web applications. In *Proceedings of the 47th Annual Southeast Regional Conference*, page 66. ACM.

Qian, K., Sztipanovits, M., and Fu, X. (2008). Automated Testing and Smart Tutoring System for Web Application. *2008 International Workshop on Education Technology and Training & 2008 International Workshop on Geoscience and Remote Sensing*, pages 582–585.

Ran, L., Dyreson, C., Andrews, A., Bryce, R., and Mallery, C. (2009). Building test cases and oracles to automate the testing of web database applications. *Information and Software Technology*, **51**(2), 460–477.

Razali, N. and Wah, Y. B. (2011). Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of Statistical Modeling and Analytics*, **2**(1), 21–33.

Ricca, F. and Tonella, P. (2001). Analysis and testing of web applications. In *Proceedings of the 23rd international conference on Software engineering*, pages 25–34. IEEE Computer Society.

Sampath, S., Bryce, R., Jain, S., and Manchester, S. (2011). A tool for combination-based prioritization and reduction of user-session-based test suites. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, number c, pages 574–577. IEEE.

Sauvé, J., Abath Neto, O., and Cirne, W. (2006). EasyAccept: a tool to easily create, run and drive development with automated acceptance tests. In *Proceedings of the 2006 international workshop on Automation of software test*, pages 111–117. ACM.

Schwarz, C., Skytteren, S., and Ø vstetun, T. (2005). AutAT: an eclipse plugin for automatic acceptance testing of web applications. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 182–183. ACM.

Silva, J., Campos, J., and Paiva, A. (2008). Model-based user interface testing with spec explorer and concurtasktrees. *Electronic Notes in Theoretical Computer Science*, **208**, 77–93.

smartesting (2012). Smartesting certifyit? 5.2. http://www.smartesting.com/index.php/cms/en /product/certify-it, acessado em 16/01/2012.

Soares, L. F. G. and Rodrigues (2006). Produção de Conteúdo Declarativo para TV Digital. *SemiSH*.

Soares, L. F. G., Rodrigues, R. F., and Moreno, M. F. (2007). Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System. *JBCS*.

Soares, S. (2004). *An aspect-oriented implementation method*. Ph.D. thesis, Universidade Federal de Pernambuco.

Somé, S. (2006). Supporting use case based requirements engineering. *Information and Software Technology*, **48**(1), 43–58.

Somé, S. and Cheng, X. (2008). An approach for supporting system-level test scenarios generation from textual use cases. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 724–729. ACM.

Thibault, S. A., Marlet, R., and Consel, C. (1999). Domain-specific languages: From design to implementation application to video device drivers generation. *IEEE Trans. Softw. Eng.*, **25**, 363–377.

Trochim, W. M. (2006). Research methods knowledge base. http://www.socialresearchmethods.net, acessado em 30/06/2012.

Turner, D. a., Park, M., Kim, J., and Chae, J. (2008). An Automated Test Code Generation Method for Web Applications using Activity Oriented Approach. *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 411–414.

W3C (2012). Css overview. http://www.w3.org/Style/CSS/Overview.html, acessado em 30/01/2012.

Xu, L. and Xu, B. (2007). Applying Agent into Intelligent Web Application Testing. *2007 International Conference on Cyberworlds (CW'07)*, **3**, 61–65.

Zhu, B., Miao, H., Zeng, H., and Chen, S. (2009). Generating Test Case from Functional Requirement of Web Applications. *2009 Second International Symposium on Electronic Commerce and Security*, pages 465–468.

Apêndices

A

Script de Teste

O cenário representa uma adaptação do cenário real de Cadastro de Aluno como forma de abstrair dependências com complexidade de configuração de ambientes e infraestrutura. No entanto, não diminui a sua validade porque o mesmo não depende de configurações específicas de um ambiente.

O cenário é auto descrito, há uma tabela contendo os elementos gráficos do formulário de teste (Tabela A.2). A Tabela A.1 apresenta parcialmente um *script* que remete ao cenário no formato do Selenium IDE.

Tabela A.1 *Script* de Cadastro de Aluno

Comando	Alvo	Valor
open	/aluno/add.html	
click	id=salvar	
verifyText	"Nome Completo é obrigatório!"	
type	id=name	Joa*o da Sil^va
click	id=send	
verifyText	"Nome Completo contém caracteres inválidos!"	
type	id=name	Joao da Silva
click	id=send	
verifyText	"Apelido é obrigatório!"	
type	id=login	joao~
click	id=send	
verifyText	"Login contém caracteres inválidos!"	
...	...	...

A.1 Elementos Gráficos do Caso de Teste de Cadastro de Aluno

Tabela A.2 Formulário de Cadastro de Aluno

Label	Representação Gráfica	Ação
Nome Completo		type
Apelido		type
Email		type
Senha		type
Confirmação de Senha		type
Sexo		click
Data de Nascimento		select
Cidade		select
Escola		select
Ensino		select
Série		select
Salvar		click
Mensagem de Texto	Texto	verifyText

B

Fonte de Dados para Variação de Dados no Cenário de Teste

O cenário utilizado, apresentado no Apêndice A, os quais são tratados em termos de variação de dados. Portanto, a seguir é apresentada a planilha de dados para a implementação de variação tanto para o método atual quanto para o método proposto guiado pelo *checklist* no Apêndice C.

O número de variações definidas para o cenário ficou fixada em cinco (5), pois ficou acordado entre a equipe que seria um valor adequado, que poderia cobrir alguns casos que são comuns, como a inserção de caracteres inválidos no início e fim de um nome, ou mesmo uma acentuação solta (sem acompanhar uma letra), ou casos como incompletude de um e-mail ao se digitar, além é claro da inserção de caracteres inválidos para e-mail como acentuação.

O valor compreende uma possível baixa cobertura para os cenários, porém deve-se levar em consideração o custo de reprodução dos testes, caso este valor for incrementado haverá um reflexo na produtividade. Um exemplo: se a equipe possui 10 casos de teste na pilha, e para cada caso há 5 elementos onde haverá injeção de variabilidade e a equipe decide uma variação de 5, ao final serão 250 execuções de testes. Caso a equipe decida que 7 é um número melhor para a variação, haverá um aumento de 100 execuções.

B.1 Fonte de Dados para o Cenário

Tabela B.1 Fonte de Dados para o Cenário

name	login	email	pass	pass_conf
$Maria# de Jesus	maria!	maria@	j3sus	j3sus
Maria de Jesus	maria	maria@domain.com	jessuss	jessuss
Jose O;telo‘	j’;se	jose@.com	12	123
Jose Otelo	jose	jose‘@domain.com	jotelo	jotelo
Faus*tus Si[l\	12j20	fausto@domain	123	123
Faustus da Silva	fausto	fausto@domain.com	ffa	ffa
J^o S0vares/	jô	jô@açucar.com	ç&a	42a
Jô Sovares	jo	jo@acucar.com	acucar	acucar
3Caçula J%	caçula	caçula@teste	caculajr	cacula
Caçula Junior	cacula	cacula@teste.com	caculajr	caculajr

C

Checklists do Experimento com o Cenário de Teste

C.1 *Checklist* de Atividades para a realização do cenário

Tabela C.1 *Checklist* de Atividades para a realização do cenário (parte 1)

Atividade	
Substituir a 1ª ocorrência de Nome Completo (id=name) no *script* pela linha 2 da fonte	
Substituir a 2ª ocorrência de Nome Completo (id=name) no *script* pela linha 3 da fonte	
Substituir a 1ª ocorrência de Apelido (id=login) no *script* pela linha 2 da fonte	
Substituir a 2ª ocorrência de Apelido (id=login) no *script* pela linha 3 da fonte	
Substituir a 1ª ocorrência de Email (id=email) no *script* pela linha 2 da fonte	
Substituir a 2ª ocorrência de Email (id=email) no *script* pela linha 3 da fonte	
Substituir a 1ª ocorrência de Senha (id=pass) no *script* pela linha 2 da fonte	
Substituir a 2ª ocorrência de Senha (id=pass) no *script* pela linha 3 da fonte	
Substituir a 1ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 2 da fonte	
Substituir a 2ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 3 da fonte	
Substituir a 1ª ocorrência de Nome Completo (id=name) no *script* pela linha 4 da fonte	
Substituir a 2ª ocorrência de Nome Completo (id=name) no *script* pela linha 5 da fonte	
Substituir a 1ª ocorrência de Apelido (id=login) no *script* pela linha 4 da fonte	
Substituir a 2ª ocorrência de Apelido (id=login) no *script* pela linha 5 da fonte	
Substituir a 1ª ocorrência de Email (id=email) no *script* pela linha 4 da fonte	
Substituir a 2ª ocorrência de Email (id=email) no *script* pela linha 5 da fonte	
Substituir a 1ª ocorrência de Senha (id=pass) no *script* pela linha 4 da fonte	
Substituir a 2ª ocorrência de Senha (id=pass) no *script* pela linha 5 da fonte	

Tabela C.2 *Checklist* de Atividades para a realização do cenário (parte 2)

Atividade	
Substituir a 1ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 4 da fonte	
Substituir a 2ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 5 da fonte	
Substituir a 1ª ocorrência de Nome Completo (id=name) no *script* pela linha 6 da fonte	
Substituir a 2ª ocorrência de Nome Completo (id=name) no *script* pela linha 7 da fonte	
Substituir a 1ª ocorrência de Apelido (id=login) no *script* pela linha 6 da fonte	
Substituir a 2ª ocorrência de Apelido (id=login) no *script* pela linha 7 da fonte	
Substituir a 1ª ocorrência de Email (id=email) no *script* pela linha 6 da fonte	
Substituir a 2ª ocorrência de Email (id=email) no *script* pela linha 7 da fonte	
Substituir a 1ª ocorrência de Senha (id=pass) no *script* pela linha 6 da fonte	
Substituir a 2ª ocorrência de Senha (id=pass) no *script* pela linha 7 da fonte	
Substituir a 1ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 6 da fonte	
Substituir a 2ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 7 da fonte	
Substituir a 1ª ocorrência de Nome Completo (id=name) no *script* pela linha 8 da fonte	
Substituir a 2ª ocorrência de Nome Completo (id=name) no *script* pela linha 9 da fonte	
Substituir a 1ª ocorrência de Apelido (id=login) no *script* pela linha 8 da fonte	
Substituir a 2ª ocorrência de Apelido (id=login) no *script* pela linha 9 da fonte	
Substituir a 1ª ocorrência de Email (id=email) no *script* pela linha 8 da fonte	
Substituir a 2ª ocorrência de Email (id=email) no *script* pela linha 9 da fonte	
Substituir a 1ª ocorrência de Senha (id=pass) no *script* pela linha 8 da fonte	
Substituir a 2ª ocorrência de Senha (id=pass) no *script* pela linha 9 da fonte	
Substituir a 1ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 8 da fonte	
Substituir a 2ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 9 da fonte	
Substituir a 1ª ocorrência de Nome Completo (id=name) no *script* pela linha 10 da fonte	
Substituir a 2ª ocorrência de Nome Completo (id=name) no *script* pela linha 11 da fonte	
Substituir a 1ª ocorrência de Apelido (id=login) no *script* pela linha 10 da fonte	
Substituir a 2ª ocorrência de Apelido (id=login) no *script* pela linha 11 da fonte	
Substituir a 1ª ocorrência de Email (id=email) no *script* pela linha 10 da fonte	
Substituir a 2ª ocorrência de Email (id=email) no *script* pela linha 11 da fonte	
Substituir a 1ª ocorrência de Senha (id=pass) no *script* pela linha 10 da fonte	
Substituir a 2ª ocorrência de Senha (id=pass) no *script* pela linha 11 da fonte	
Substituir a 1ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 10 da fonte	
Substituir a 2ª ocorrência de Confirmação de Senha (id=pass_conf) no *script* pela linha 11 da fonte	

D

Instrumentos do Experimento

D.1 Planilha de Coleta de Dados

Tabela D.1 Planilha de Coleta de Dados

ID =	TCS (min)
Método Atual	
Método Proposto	
...	...
ID =	TCS (min)
Método Atual	
Método Proposto	
...	...
ID =	TCS (min)
Método Atual	
Método Proposto	

D.2 Questionário para Análise Qualitativa

Questionário para Análise Qualitativa
Numa escala de 1 a 10, no qual 1 significa não útil e 10 significa muito útil, como você avalia a utilidade do método proposto para a criação e execução de suítes de teste com variação de dados?
Numa escala de 1 a 10, no qual 1 significa muito fácil e 10 significa muito difícil, como você avalia a dificuldade na utilização do método proposto para a criação e execução de suítes de teste com variação de dados em relação ao método atual?
Por favor, escreva alguma sugestão que você acredita ser útil para a melhoria do método.

Tabela D.2 Questionário para Análise Qualitativa.

E

Experimento Piloto

O experimento piloto serve como treinamento para a realização do experimento propriamente dito. Os elementos componentes do piloto são os mesmos do experimento, tomando-se o cuidado para não serem muito semelhantes para não incorrer num maior viés. O cenário é o que pode ser visto na Tabela E.1. Possui um *script* semelhante ao que pode ser visto no Apêndice A. Utiliza uma fonte de dados, observável na Tabela E.2. Como guia para execução do piloto é possível derivar da do experimento, como no Apêndice C, fazendo o mapeamento para o formulário do piloto.

E.1 Elementos Gráficos do Caso de Teste de Cadastro de Professor

Tabela E.1 Formulário de Cadastro de Professor

Label	Representação Gráfica	Ação
Nome Completo		type
Apelido		type
Escola		select
Ensino		type
Cidade		select
Senha		type
Confirmação de Senha		type
Sexo		click
Salvar		click
Mensagem de Texto	Texto	verifyText

E.2 Fonte de Dados para o Cenário

Tabela E.2 Fonte de Dados para o Cenário

name	login	email	pass	pass_conf
$Augusto# Miranda	augusto!	augusto@	augus70	augus70
Augusto Miranda	augusto	augusto@domain.com	augustomir	augustomir
Saulo R;odrigues‘	sa’;ulo	saulo@.com	12	123
Saulo Rodrigues	saulo	saulo‘@domain.com	saurodr	saurodr
Adoni*do Per[eira\	12p10	adonildo@domain	092	092
Adonildo Pereira	adonildo	adonildo@domain.com	peradon	peradon
Crisˆostomos Al9vares/	aˆlvares	aˆlvares@alváres.com	á&bv8a	920i
Crisostomos Alvares	alvares	crisostomos@alvares.com	alva	alva
8Otamano Fil%ho	o8omano	o8omano@tes%te	otomano	otomanf
Otamano Filho	otomano	otomano@teste.com	otomanf	otomanf

F

Suítes de Exemplo

A seguir serão apresentados dois exemplos de suítes de testes com variação de entrada de dados utilizando a fonte descrita no Apêndice B.

F.1 Método Atual

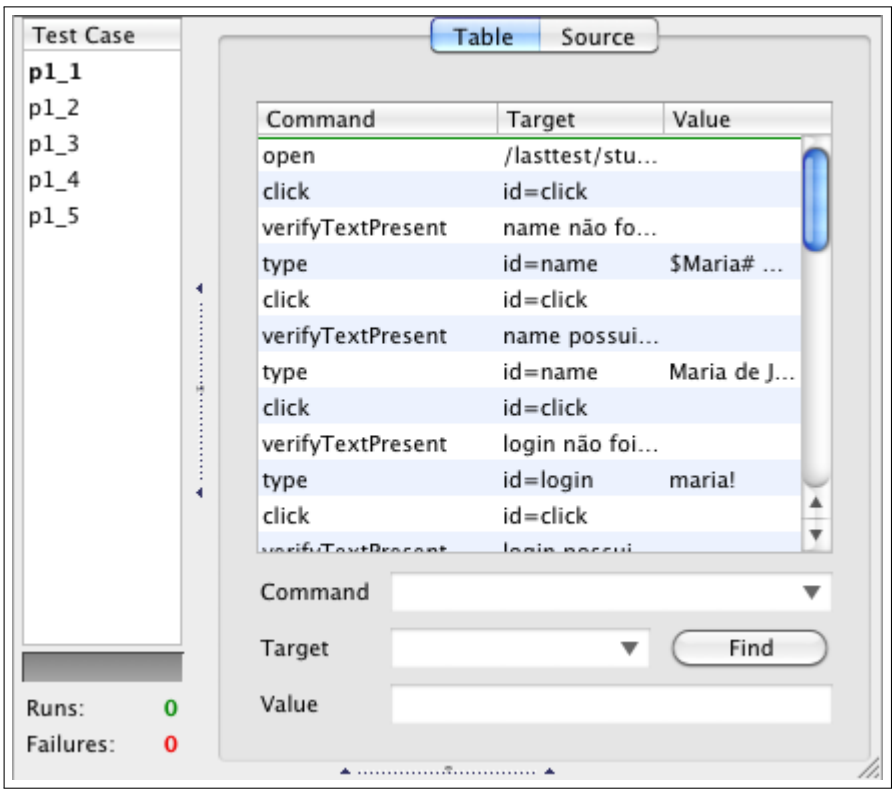

Figura F.1 Suíte de Exemplo do Método Atual para cadastro de Estudante

F.2 Método Proposto

```
suite student_registering
    script := @"C:\\temp\\scripts\\proposed2\\Student.sel";
    source := $"C:\\temp\\scripts\\proposed2\\Student.csv";

    run script fetching {
                        source [0] : nome [0] , login [0], email [0],
                             pass [0], pass_conf [0].
                        source [1] : nome [1] , login [1], email [1],
                             pass [1], pass_conf [1]
                        };
    run script fetching {
                        source [2] : nome [0] , login [0], email [0],
                            pass [0], pass_conf [0].
                        source [3] : nome [1] , login [1], email [1],
                            pass [1], pass_conf [1]
                        };

    run script fetching {
                        source [4] : nome [0] , login [0], email [0],
                            pass [0], pass_conf [0].
                        source [5] : nome [1] , login [1], email [1],
                            pass [1], pass_conf [1]
                        };

    run script fetching {
                        source [6] : nome [0] , login [0], email [0],
                            pass [0], pass_conf [0].
                        source [7] : nome [1] , login [1], email [1],
                            pass [1], pass_conf [1]
                        };

    run script fetching {
                        source [8] : nome [0] , login [0], email [0],
                            pass [0], pass_conf [0].
                        source [9] : nome [1] , login [1], email [1],
                            pass [1], pass_conf [1]
                        };
end
```

Figura F.2 Suíte de Exemplo do Método Proposto para cadastro de Estudante