PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Silvia Regina Assis Meireles

EVOLUÇÃO DA FERRAMENTA *WEB GUITAR* PARA GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE DE INTERFACE PARA APLICAÇÕES WEB

Manaus
2015

SILVIA REGINA ASSIS MEIRELES

EVOLUÇÃO DA FERRAMENTA *WEB GUITAR* PARA GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE DE INTERFACE PARA APLICAÇÕES WEB

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas, como parte dos requisitos necessários para a obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Arilo Cláudio Dias Neto

Manaus
2015

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

M514e

Meireles, Silvia Regina Assis
Evolução da ferramenta Web Guitar para geração automática de casos de teste de interface para Aplicações Web / Silvia Regina Assis Meireles. 2015
103 f.: il. color; 29,7 cm.

Orientador: Arilo Claudio Dias Neto
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Aplicações Web. 2. Teste Baseado em Modelos. 3. Automação de teste. 4. Teste GUI. 5. Geração de casos de teste. I. Dias Neto, Arilo Claudio II. Universidade Federal do Amazonas III. Título

SILVIA REGINA ASSIS MEIRELES

EVOLUÇÃO DA FERRAMENTA *WEB GUITAR* PARA GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE DE INTERFACE PARA APLICAÇÕES WEB

Proposta de Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas, como parte dos requisitos necessários para a obtenção do título de Mestre em Informática.

Banca Examinadora

_____

Prof. Dr. Arilo Cláudio Dias Neto (Orientador)
Universidade Federal do Amazonas (UFAM)

_____

Prof. Dr. Auri Marcelo Rizzo Vincenzi
Universidade Federal de Goiás (UFG)

_____

Prof. Dr. Raimundo da Silva Barreto
Universidade Federal do Amazonas (UFAM)

Manaus
2015

À Deus, ao meu esposo, aos meus familiares e amigos pelo carinho e apoio em todos os momentos.

Agradecimentos

À Deus, por seu grande amor a ponto de ter enviado seu único filho para morrer por nós; por suas misericórdias que se renovam a cada manhã; por ser um Pai maravilhoso e por me permitir chegar até aqui. “Sabemos que Deus age em todas as coisas para o bem daqueles que amam a Deus, dos que foram chamados de acordo com o seu propósito” (Romanos 8:28).

Ao meu esposo, Carlos, pelo amor, cumplicidade, dedicação, incentivo, revisão do texto e compreensão em todos os momentos, principalmente nos mais difíceis.

Aos meus avós, Jardelina e César (que só vou reencontrar quando Jesus voltar), pelo amor incondicional, por investirem e acreditarem em mim e por serem meus exemplos de vida. Aos meus queridos pais, Nete e Danival, pelo amor e carinho. Aos meus irmãos e tios, Andrisson, Geane, Johmara, Danilo, Neto, Daniel, Midian, Elias, Celeste, Carol, Augusto, Luiza e Carlson, pela torcida e conselhos. À todos meus queridos primos e como a lista tende ao infinito, vou citar os mais próximos, Tiago e Kalyne, pelo cuidado e incentivo.

Aos meus sogros, Naíde e Carlos, pela dedicação e carinho. Aos meus cunhados pela amizade e consideração e aos meus sobrinhos pelo imenso carinho.

Ao meu orientador, professor Arilo Cláudio, pelos ensinamentos, paciência, incentivo e por sempre me indicar uma solução prática quando todas minhas ideias já se esgotaram, a ponto de eu pensar, “como não pensei nisso antes?”

Aos meus amigos e amados irmãos em Cristo pelas palavras sábias, especialmente aos amigos do meu grupo de oração, que me apoiam com suas orações e torcem por mim, Vivian, Francisco, Dani, Jailton, Mirthis, Ney, Alessandra, Rafa e Francisco Sales.

Aos integrantes do grupo ExperTS pela troca de ideias e experiências, pela cooperação e por tornarem nosso laboratório tão harmonioso, descontraído. Aqui, não posso deixar de destacar Renata, minha fiel escudeira, por me ajudar em todos os momentos; Kariny pela revisão do texto e ajuda com os diagramas; Karina, Awdren, Jonathas, Rodrigo, Luana, Anderson, Allan, Thaynã, Nayane, Jeanne, Aurélio, Bruna, Kirmayr, Urique, Laíza, Juliana, Ingrid, Erick, Larissa e Jade pelo

incentivo e pelas excelentes dicas e Odette pela ajuda com o mapeamento sistemático. Aos amigos de outros laboratórios do ICOMP pelas palavras de incentivo, troca de experiências e contribuição com este trabalho, Rainer, Herbert, Eliza, Will, Rosana, Josi e Lídia.

Aos professores do ICOMP com quem estudei, especialmente a professora Tayana Conte, pela motivação e valiosas dicas. Também aos funcionários da secretaria do ICOMP pela agilidade no atendimento.

Ao INdT que tem nos proporcionado ótimas oportunidades de crescimento profissional por meio da parceria com a UFAM.

Ao professor Edward Moreno pelo incentivo desde a graduação.

Aos meus amigos pelo apoio e incentivo, Sal, Ageu, Mila, Walda, Val, Paty, Pri, Helena, Paulo e Deolinda que já conhece toda minha pesquisa de tanto revisar meu texto. E, aos meus amigos que mesmo distantes têm um lugar guardado no meu coração.

A CAPES pelo apoio financeiro.

E a todos que me ajudaram direta ou indiretamente.

Resumo da Dissertação apresentada à UFAM/AM como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática (M.Sc.)

EVOLUÇÃO DA FERRAMENTA WEB GUITAR PARA GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE DE INTERFACE PARA APLICAÇÕES WEB

Silvia Regina Assis Meireles

Março / 2015

Orientador: Prof. Dr. Arilo Cláudio Dias Neto

Aplicações Web estão presente em nossa rotina diária e são fundamentais em diversas áreas, tais como educação, saúde e entretenimento. Uma falha nessas aplicações pode ocasionar grandes perdas, portanto é essencial garantir a qualidade das mesmas. Uma forma de se alcançar esse objetivo é por meio do teste de software. Embora o teste de software traga inúmeros benefícios, ele acrescenta elevados custos ao projeto de desenvolvimento, que devem ser reduzidos. Uma estratégia utilizada para isso é a automação de teste. Atividades repetitivas e propensas a enganos são fortes candidatas a serem automatizadas, dentre elas podemos citar a geração de casos de teste.

A ferramenta *Web Guitar* possibilita a geração e execução dos casos de teste, a partir do modelo estrutural da aplicação Web testada. Esse modelo apresenta problemas quanto a sua precisão. Assim, neste trabalho propõe-se sua evolução com o objetivo de aumentar o número de casos de teste gerados e melhorar a sua qualidade. Como resultado, foi implementada uma nova versão desta ferramenta que varre uma aplicação web identificando novos elementos de interface e visitando diferentes instâncias de uma mesma página com o intuito de prover um modelo mais completo.

Os resultados do estudo de caso realizado com duas aplicações Web, Sistema de Apoio ao PPGI/UFAM e Disk Transito, forneceram indícios da viabilidade deste trabalho, visto que a ferramenta proposta (*WG-Modificada*) contribuiu para a geração de um número maior de casos de teste para o Sistema de Apoio ao PPGI/UFAM, apesar do seu tempo de execução ser superior ao tempo obtido pela versão original da ferramenta. Ao se analisar o tempo necessário para gerar cada caso de teste, verifica-se que a *WG-Modificada* foi mais eficiente que a versão original. Porém, os resultados do estudo para o Disk Transito evidenciaram problemas e limitações que serão tratados em trabalhos futuros.

Abstract of Thesis presented to UFAM/AM as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

EVOLUTION OF WEB GUITAR TOOL FOR AUTOMATIC GENERATION OF INTERFACE TEST CASES FOR WEB APPLICATIONS

Silvia Regina Assis Meireles

March / 2015

Advisor: Arilo Cláudio Dias Neto

Web applications are in our daily routine and are fundamental in several areas, such as education, health, and entertainment. A failure in these applications can cause financial loss, so it is essential to ensure their quality. One way to achieve this goal is through software testing. Although the testing activities bring many benefits, they add cost to a software development project, which should be reduced. A strategy used for this is testing automation. Repetitive and error prone activities are strong candidates to be automated, among them we can mention the test cases generation.

The Web Guitar tool enables the generation and execution of test cases from the structural model of Web application under test. This model presents problems related to its accuracy. Thus, this work proposes its evolution aiming to increase the number of generated test cases and improve their quality. As contribution, a new version of Web Guitar tool was implemented to sweep a web application identifying new GUI elements and visiting different instances of a same page in order to provide a more complete model.

The results obtained in a case study performed with two Web applications, System to Support PPGI/UFAM and DiskTransito, provided evidence of the feasibility of this work, since the proposed tool (*WG-Modified*) contributed to the generation of a greater number of test cases for System to Support PPGI/UFAM, despite its execution time exceed the time calculated for the original tool. When analyzing the time required to generate each test case, *WG-Modified* was more efficient than its original version. However, the results of this study for the second application (DiskTransito) showed some problems and limitations that will be addressed in future works.

ÍNDICE

ÍNDICE DE FIGURAS XIV
ÍNDICE DE TABELAS XVII
CAPÍTULO 1 - INTRODUÇÃO 1
1.1. Contextualização e Motivação 1
1.2. Descrição do Problema 2
1.3. Hipótese 4
1.4. Objetivos 4
1.4.1. Objetivo geral 4
1.4.2. Objetivos específicos 4
1.5. Metodologia da Pesquisa 5
1.5.1. Fase de Concepção 5
1.5.2. Fase de Avaliação 6
1.6. Estrutura do Documento 7
CAPÍTULO 2 - REFERENCIAL TEÓRICO 8
2.1. Aplicações Web 8
2.2. Teste de Software 10
2.2.1. Teste de Interface Gráfica do Usuário 10
2.2.1.1. Caso de Teste GUI 11
2.2.1.2. Oráculo de Teste GUI 11
2.2.2. Automação de Teste 12
2.2.3. Teste Baseado em Modelos 12
2.4. Trabalhos relacionados 14
2.5. Web Guitar 15
2.5.1. Modelo GUI 16
2.5.2. Fluxo de Trabalho da Web Guitar 17
2.5.3. Conceitos relacionados a execução da Web Guitar 18
2.5.3.1. Entradas manuais 18
2.5.3.2. Largura e Profundidade 18
2.5.3.3. Árvore GUI e Grafo EFG 20
2.5.4. Execução da Web Guitar 20
2.6. Estrutura Web Guitar 24

2.7. Mapeamento Sistemático sobre Técnicas de TBM para Aplicações Web 26
2.7.1. Definição do Objetivo e Questões de Pesquisa 26
2.7.2. Fontes de Pesquisa e String de busca 27
2.7.3. Critérios de inclusão dos artigos 28
2.7.4. Formulário de Extração de Dados 29
2.7.5. Análise dos Resultados 35
2.7.5.1. Ano de publicação 35
2.7.5.2. Local de publicação 35
2.7.5.3. Tipo de contribuição 36
2.7.5.4. Tipo de estudo 36
2.7.5.5. Localização do teste 37
2.7.5.6. Modelos utilizados 37
2.7.5.7. Nível de teste 38
2.7.5.8. Características de qualidade 39
2.7.5.9. Existência de Apoio Ferramental 40
2.7 Considerações Finais 40
CAPÍTULO 3 – EVOLUÇÃO DA FERRAMENTA WEB *GUITAR* 42
3.1. Introdução 42
3.2. Atividades realizadas 42
3.2.1. Identificar limitações descritas na literatura 43
3.2.2. Utilização da ferramenta 43
3.2.3. Priorizar modificações 45
3.3. Modificações realizadas 46
3.3.1. Valores de entrada 46
3.3.2. Inclusão de várias instâncias 54
3.3.3. Inclusão de número máximo de visitas 57
3.3.4. Outras modificações 58
3.4. Limitações identificadas e não tratadas 59
CAPÍTULO 4 - ESTUDO DE VIABILIDADE PARA AVALIAÇÃO DA CUSTOMIZAÇÃO DA *WEB GUITAR* 61
4.1. Introdução 61
4.2. Definição do Estudo de Viabilidade 61

4.2.1. Propósito 61
4.2.2. Perspectiva 62
4.2.3. Questões e Métricas 62
4.3. Planejamento do Estudo 63
4.3.1. Formulação de Hipóteses 63
4.3.2. Projetos a serem testados 65
4.4. Execução do Estudo 67
4.5. Resultado do Estudo de Viabilidade 70
4.5.1. Resultados Sistema de Apoio PPGI/UFAM 71
4.5.2. Resultados Disk Transito 79
4.6. Ameaças à Validade do Estudo 88
4.6.1. Validade Interna 88
4.6.2. Validade Externa 89
4.6.3. Validade de Construção 89
4.6.4. Validade de Conclusão 89
4.7. Considerações Finais 90
CAPÍTULO 5 – CONCLUSÃO 91
5.1. Considerações Finais 91
5.2. Contribuições 91
5.3. Limitações 92
5.4. Trabalhos futuros 92
REFERÊNCIAS BIBLIOGRÁFICAS 93

LISTA DE SIGLAS E ABREVIATURAS

AJAX – *Asynchronous Javascript and XML*

AUT – *Aplication Under Test*

CT – Caso de Teste

DOM – *Dynamic Document Object*

EFG – *Event Flow Graph*

GQM – *Goal/Question/Metric*

GUI – *Grafical User Interface*

HTML – *Hyper Text Markup Language*

MEF – Máquina de Estados Finitos

MS- Mapeamento Sistemático

PHP – *Hypertext Preprocessor*

PICO – *Population, Intervention, Comparison and Outcomes*

RIAs – *Rich Internet Applications*

TBM – Teste Baseado em Modelos

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

XML – *Extensible Markup Language*

W3C – *World Wide Web Consortium*

ÍNDICE DE FIGURAS

Figura 1: Relação entre as ferramentas *Guitar* e *Web Guitar* 3
Figura 2: Metodologia de Pesquisa adotada 5
Figura 3: Categorias de Aplicações Web (ISABEL, 2011) 9
Figura 4: Atividades de TBM (DIAS-NETO, 2009) 13
Figura 5: Modelo da GUI 16
Figura 6: Fluxo de trabalho da ferramenta Web *Guitar* – (Adaptado de NGUYEN *et al.*, 2013) 17
Figura 7: Componentes terminais 19
Figura 8: Página index do MTCONTROOL 19
Figura 9: Página index do SIS3 20
Figura 10: Comando para executar o *Web Ripper* 20
Figura 11: Página inicial para executar o *Web Ripper* 21
Figura 12: Parte da árvore GUI referente a página index do SIS3 21
Figura 13: Comando para executar o Conversor de grafo 22
Figura 14: Parte do arquivo EFG 22
Figura 15: Comando para executar o Gerador de casos de teste 23
Figura 16: Casos de teste gerados 23
Figura 17: Comando para executar o Web Replayer 23
Figura 18: Parte do arquivo de estado gerado 24
Figura 19: Diagrama de Classes Web Guitar 25
Figura 20: Ano de publicação 35
Figura 21: Tipo de contribuição 36
Figura 22: Tipo de Estudo 37
Figura 23: Localização dos testes 37
Figura 24: Modelos utilizados 38
Figura 25: Níveis de teste 39
Figura 26: Características de qualidade 39
Figura 27: Gera ferramentas 40
Figura 28: Página index.php (1) do *Phone Book* 44
Figura 29: Página index.php (2) do *Phone Book* 44

Figura 30: Web Guitar modificada 46
Figura 31: Página *create.php*.....47
Figura 32: Parte do arquivo *configurationOutput.xml* referente a página *create.php* 48
Figura 33: Arquivo *configurationOutput.xml* da página *create.php* editado.....49
Figura 34: Página *create.php* preenchida.....50
Figura 35: Página Nova Ocorrência – Instância 1 51
Figura 36: Página Nova Ocorrência – Instância 2 52
Figura 37: Modificações realizadas no código para inserir valores de entrada 53
Figura 38: Instância 1 da página Dados do Aluno 54
Figura 39: Instância 2 da página Dados do Aluno 55
Figura 40: Chamada ao *Web Ripper* com inserção do parâmetro de número de instâncias (*-i*) e número de visitas (*-v*).....56
Figura 41: Modificações realizadas no código para trabalhar com várias instâncias 57
Figura 42: Modificação realizada no código para se incluir o número máximo de visitas 58
Figura 43: Página main.html.....59
Figura 44: Componentes *Autocomplete* (esquerda) e *PickList* (direita).....59
Figura 45: Métricas do Sistema de Apoio ao PPGI/UFAM.....65
Figura 46: Métricas do Disk Transito 66
Figura 47: Ambiente de execução do experimento.....67
Figura 48: Definição dos valores de largura e profundidade.....68
Figura 49: Execução do Web Ripper 69
Figura 50: Execução do Experimento.....70
Figura 51: Gráfico de Casos de Teste gerados PPGI/UFAM (d=3) 72
Figura 52: Gráfico de Tempo/Caso de Teste PPGI/UFAM (d=3).....72
Figura 53: Gráfico de Casos de Teste gerados PPGI/UFAM (d=5) 73
Figura 54: Gráfico de Tempo/Caso de Teste PPGI/UFAM (d=5).....74
Figura 55: Gráfico de casos de Teste gerados PPGI/UFAM (d=8).....75
Figura 56: Página Gerenciar Alunos PPGI 75
Figura 57: Gráfico de Tempo/Caso de Teste PPGI/UFAM (d=8).....76

Figura 58: Gráfico de casos de teste gerados PPGI/UFAM (d=13) 77
Figura 59: Gráfico de Tempo/Caso de Teste PPGI/UFAM (d=13) 77
Figura 60: Gráfico de casos de teste gerados Disk Transito (d=3) 80
Figura 61: Gráfico de Tempo/Caso de Teste Disk Transito (d=3) 81
Figura 62: Gráfico de casos de teste gerados Disk Transito (d=5) 82
Figura 63: Página de Cadastros básicos 83
Figura 64: Gráfico de Tempo/Caso de Teste Disk Transito (d=5) 83
Figura 65: Gráfico de casos de teste gerados Disk Transito (d=8) 84
Figura 66: Gráfico de Tempo/Caso de Teste Disk Transito (d=8) 85
Figura 67: Gráfico de casos de teste gerados Disk Transito (d=13) 86
Figura 68: Gráfico de Tempo/Caso de Teste Disk Transito (d=13) 86

ÍNDICE DE TABELAS

Tabela 1: Objetivo segundo o paradigma GQM.....27
Tabela 2: Campos extraídos no mapeamento sistemático 29
Tabela 3: Caracterização dos Trabalhos Selecionados no Mapeamento Sistemático.....31
Tabela 4: Principais locais de publicação 35
Tabela 5: Resumo de métricas do Sonarqube para os sistemas a serem testados.....66
Tabela 6: Resultado PPGI/UFAM para profundidade (d=3).....71
Tabela 7: Resultado PPGI/UFAM para profundidade (d=5).....73
Tabela 8: Resultado PPGI/UFAM para profundidade (d=8).....74
Tabela 9: Resultado PPGI/UFAM para profundidade (d=13).....76
Tabela 10: Resultados do Sistema de Apoio ao PPGI/UFAM 78
Tabela 11: Resultado Disk Transito para profundidade (d=3) 80
Tabela 12: Resultado Disk Transito para profundidade (d=5) 81
Tabela 13: Resultado Disk Transito para profundidade (d=8) 84
Tabela 14: Resultado Disk Transito para profundidade (d=13) 85
Tabela 15: Resultados do Disk Transito 87

CAPÍTULO 1 - INTRODUÇÃO

Neste capítulo serão apresentados o contexto, a motivação e o problema que será tratado neste trabalho. Também serão apresentados a hipótese, os objetivos, a metodologia de pesquisa adotada e a organização deste trabalho.

1.1. Contextualização e Motivação

Sistemas web (ou aplicações web) tornaram-se simplesmente necessários e fundamentais em diversas áreas. Uma prova disto é que se está cada vez mais dependente desse tipo de aplicação, seja para acessar uma conta bancária ou para planejar uma viagem. Segundo KAPPEL (2004), Aplicações Web são sistemas de software baseados em tecnologias e padrões da *World Wide Web Consortium* (W3C) que fornecem recursos específicos da Web, tais como conteúdo e serviços por meio de uma interface de usuário, o navegador Web (em inglês, *web browser*).

Com a crescente demanda de aplicações Web de alta complexidade e que exigem elevado padrão de qualidade, surge também a necessidade de buscar estratégias que permitam aumentar a qualidade do produto e reduzir o custo e/ou tempo de desenvolvimento. Uma das formas de se melhorar a qualidade em sistemas de software é por meio do teste de software, incluindo Aplicações Web (DELAMARO *et al.*, 2007; FASOLINO *et. al*., 2013). JURISTO *et al.* (2006) afirmam que uma abordagem de teste bem-sucedida pode reduzir significativamente o esforço de teste e contribuir para a melhoria da qualidade do produto, aumentando a satisfação do cliente e reduzindo os custos de manutenção.

O processo de teste é tema de vários estudos, cujos resultados têm comprovado cada vez mais sua importância no projeto de software. CHERNONOZHKIN (2001) estima que a atividade de teste contribua com cerca de 50% do custo total do desenvolvimento de software. Em outro estudo, KANER (1999) define que o custo de se encontrar e corrigir defeitos nos programas varia de 40% a 80% do seu custo total de desenvolvimento.

Para MARIANI *et al.* (2012), gerar casos de teste automaticamente tanto pode reduzir os custos quanto aumentar a eficácia dos testes. Apesar de inúmeras

pesquisas demonstrarem os benefícios da automação de teste, XIAOCHUN (2008) aponta as principais barreiras para sua utilização na indústria:

- Elevado custo para passar todo o processo de concepção, desenvolvimento e execução de *scripts* de teste;
- Necessidade de pessoal com conhecimento e habilidades específicas de programação;
- Gerenciamento consistente de *scripts* de dados e casos de teste.

Gerar automaticamente casos de teste é uma tarefa difícil e complexa que desafia pesquisadores e profissionais ao longo de décadas, e ainda é um problema que continua amplamente aberto (MARIANI *et al.*, 2012). Dentre as abordagens de automação de testes de software existentes, o Teste Baseado em Modelos (TBM) se destaca por sua ampla utilização em diversos domínios, incluindo Web (GAROUSI *et al.*, 2013). TBM é uma abordagem que envolve o desenvolvimento de um modelo a partir do qual os testes são gerados. Esse modelo é essencialmente uma especificação de entradas para o software, que pode ser desenvolvido ainda no início do ciclo de desenvolvimento de software (DALAL *et al.*, 1999).

1.2. Descrição do Problema

Atualmente, a maioria das aplicações de software fornece serviços aos seus usuários por meio de uma Interface Gráfica de Usuário (do inglês, *Grafical User Interface* – GUI) que consiste de objetos virtuais (*widgets*) que tornam o software mais fácil de usar. Essa interface é um dos principais componentes do software moderno (GANOV *et al*. 2009; RAUF *et al.*, 2010). Portanto, testá-la é fundamental para garantir a qualidade de produtos de software (MIAO e YANG, 2010).

No teste GUI há uma interação direta com a aplicação simulando um usuário final. Esse tipo de teste contém uma série de eventos, tais como, campos de entradas e cliques de *mouse* que estão associados a cada interação do usuário. MEMON *et al.* (2001) e MOREIRA e PAIVA (2014) apontam a quantidade de interações possíveis dos usuários como a grande dificuldade de testar GUIs. AHO *et al.* (2011) enfatizam que o teste manual de GUIs é trabalhoso, caro e propenso a enganos. Logo, é fundamental buscar estratégias para reduzir os elevados custos do teste GUI, e uma alternativa seria a aplicação de testes automatizados. Nesse contexto, o TBM se encaixa perfeitamente, pois ele é baseado em processos

automatizados, evitando assim atividades propensas a erros (SANTOS-NETO *et al.*, 2007).

Segundo AHO *et al.* (2011), GUI Testing frAmewoRk (*Guitar)* é uma das ferramentas mais avançadas de código aberto que permite criar automaticamente modelos para TBM. Essa ferramenta possui uma extensão para aplicações Web, denominada *Web Guitar*. A relação entre as ferramentas *Guitar* e *Web Guitar* é mostrada na Figura 1:

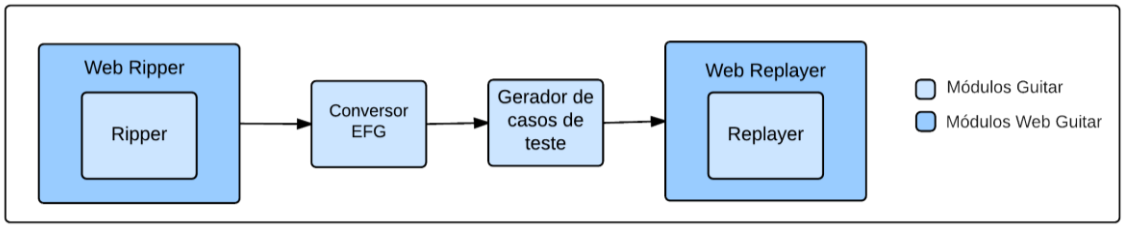

Figura 1: Relação entre as ferramentas *Guitar* e *Web Guitar*

1. *Web Ripper* é o módulo responsável por estender as funcionalidades do *Ripper* para aplicações Web;
2. Guitar:
   a) *Ripper* é o módulo responsável pela geração da árvore GUI (modelo estrutural GUI) por meio da execução da aplicação testada;
   b) Conversor de grafo é o módulo responsável pela conversão da árvore GUI em um grafo, denominado Grafo de Fluxo de Evento (do inglês, *Event Flow Graph* - EFG);
   c) Gerador de casos de teste é o módulo responsável pela geração automática de casos de teste;
   d) *Replayer* é o módulo responsável pela execução automática dos casos de teste gerados.
3. *Web Replayer* é o módulo responsável por estender as funcionalidades do *Replayer* para aplicações Web.

NGUYEN *et al.* (2013) apontam algumas limitações relacionadas ao módulo *Ripper* que são:

- *Web Ripper* possui a premissa de visitar uma única instância de cada página da aplicação Web. Desta forma, páginas que possuam

variações de conteúdo podem ter alguns componentes ignorados, levando à testes incompletos na aplicação web;

- Componentes cujas propriedades mudam em tempo de execução apresentam problemas em sua identificação, causando a geração de testes incompletos.

Esses problemas são relacionados à precisão das árvores GUI e influenciam na quantidade de casos de teste gerados e executados.

1.3. Hipótese

Baseado no contexto e problema apresentados nas seções anteriores, a hipótese definida para este trabalho considera o seguinte cenário:

> A adição de novos recursos à ferramenta *Web Guitar* permite melhorar a completude da árvore GUI gerada, contribuindo para o aumento no número de casos de teste gerados pela ferramenta.

1.4. Objetivos

Nesta seção serão apresentados os objetivos desta pesquisa.

1.4.1. Objetivo geral

Aprimorar a árvore GUI produzida pela ferramenta *Web Guitar* para aumentar o número de casos de teste gerados por meio da ampliação da busca de páginas em aplicações Web.

1.4.2. Objetivos específicos

- Fornecer um corpo de conhecimento sobre os modelos e características Web utilizados em TBM aplicado a Aplicações web;
- Evoluir o módulo Web *Ripper* da ferramenta Web *Guitar* para melhorar a árvore GUI gerada;
- Avaliar experimentalmente a evolução da ferramenta Web *Guitar* em ambientes controlados, visando aumentar a eficácia e eficiência dos casos de teste gerados.

1.5. Metodologia da Pesquisa

A metodologia de pesquisa adotada neste trabalho é baseada na metodologia proposta em MAFRA *et al.* (2006), a qual é constituída das seguintes fases: concepção e avaliação da tecnologia, conforme mostrado na Figura 2.

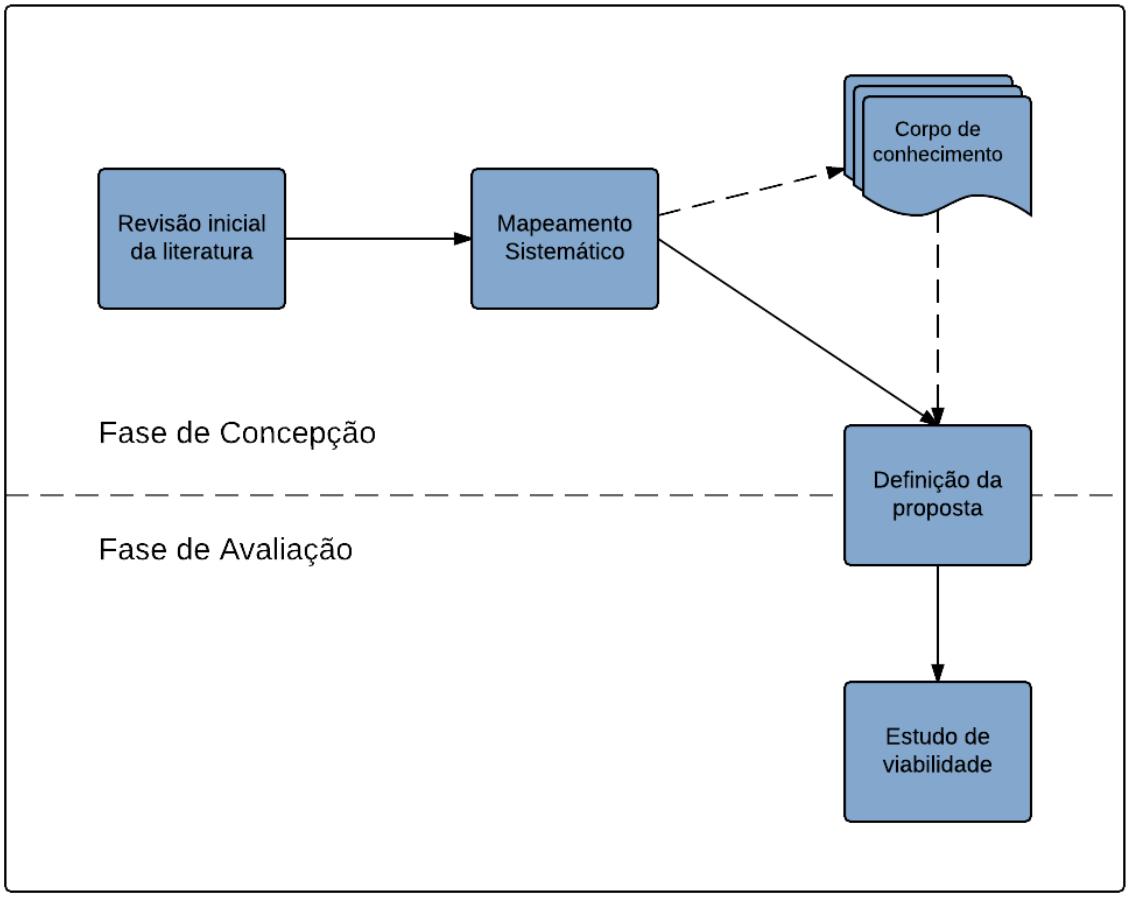

Figura 2: Metodologia de Pesquisa adotada

1.5.1. Fase de Concepção

A fase de Concepção foi baseada em SPÍNOLA *et al.* (2008) que apresenta um conjunto de atividades que utiliza estudos secundários e primários para apoiar à concepção de novas tecnologias de software. Essa fase foi conduzida conforme apresentado na Figura 2. A seguir são descritos detalhes da execução de cada atividade:

- Revisão inicial da literatura: Esta atividade foi realizada no período de Março a Outubro de 2013, na qual foi conduzida uma revisão da literatura sobre TBM e

Aplicações Web. Esta pesquisa incluiu a busca de aspectos relevantes para a caracterização de TBM no contexto de Aplicações Web.

- Mapeamento sistemático: Esta atividade foi realizada no período de Novembro de 2013 à Março de 2014 com o propósito de se obter conhecimento sobre as abordagens de TBM para aplicações Web descritas na literatura técnica. O planejamento e os resultados desse estudo serão apresentados no Capítulo 2.
- Definição da Proposta: Realizada de Abril à Dezembro de 2014, a partir do corpo de conhecimento resultante do Mapeamento Sistemático, passou-se a trabalhar na abordagem proposta. Após identificação de limitações na ferramenta *Web Guitar*, decidiu-se adaptá-la com o objetivo de reduzir as limitações identificadas e favorecer seu amadurecimento para transferi-la ao ambiente industrial. Os resultados desta etapa serão apresentados no Capítulo 3 deste trabalho.

1.5.2. Fase de Avaliação

A fase de Avaliação da tecnologia foi baseada em SHULL *et al. (*2001), cujo propósito é avaliar a tecnologia proposta desde sua definição até sua transferência para a indústria. Essa fase é composta por quatro etapas, das quais até o presente momento foi executada a primeira etapa.

- **Estudo de Viabilidade**: Este estudo foi executado de Dezembro de 2014 a Fevereiro de 2015. Uma vez realizada a adaptação da ferramenta *Web Guitar*, o próximo passo foi avaliá-la e para isso fez-se necessário planejar e executar um estudo experimental com o objetivo de caracterizar a abordagem proposta em relação à sua capacidade de gerar um conjunto de casos de teste que possibilite revelar defeitos, visando permitir a transferência dessa tecnologia para o contexto industrial. Esse estudo experimental teve como objetivo avaliar a eficiência e eficácia da ferramenta *Web Guitar* modificada em relação à ferramenta original. Os resultados deste estudo serão apresentados no Capítulo 4 deste trabalho.

1.6. Estrutura do Documento

Este trabalho está organizado como segue: No Capítulo 2, será apresentado o referencial teórico e o mapeamento sistemático realizado com o propósito de identificar e analisar os modelos adotados para representação de aplicações web. No Capítulo 3, será apresentada a adaptação da ferramenta *Web Guitar*, que é a proposta e contribuição principal desta dissertação. No Capítulo 4, serão apresentados o planejamento, execução e resultados do Estudo de Viabilidade realizado com duas aplicações Web reais. No Capítulo 5 será apresentada a Conclusão deste trabalho e trabalhos futuros que darão continuidade a esta pesquisa.

CAPÍTULO 2 - REFERENCIAL TEÓRICO

Neste capítulo serão apresentados os conceitos relacionados ao tema desta pesquisa, tais como Aplicações Web, Teste de Software, Teste GUI, TBM e o funcionamento da ferramenta *Web Guitar*. Também serão descritos o planejamento, execução e resultados do mapeamento sistemático realizado sobre TBM no contexto de Aplicações Web.

2.1. Aplicações Web

Aplicações web são sistemas de software baseados em tecnologias e padrões da *World Wide Web Consortium* (W3C) que fornecem recursos específicos da Web, tais como conteúdo e serviços por meio de uma interface de usuário, o navegador Web (KAPPEL, 2004).

Nos últimos anos, as aplicações Web têm despertado tanto o interesse de empresas quanto da comunidade acadêmica. Por isso, técnicas e métodos são propostos a fim de oferecer uma estrutura adequada às características da Web. Por essas razões, surgiu uma nova linha de pesquisa em Engenharia de Software, denominada Engenharia Web. Esta linha de pesquisa cresce a cada ano, provando que aplicações Web têm características especiais e, portanto, necessitam de um tratamento especial (ESCALONA *et al.*, 2007).

Para DI LUCCA e FASOLINO (2006), as principais características das Aplicações Web são:

- Grande número de usuários com acesso simultâneo ao sistema;
- Ambiente de execução heterogêneo, composto por diversos *hardwares*, sistemas operacionais, servidores, navegadores Web, dentre outros;
- Natureza heterogênea que depende da variedade de componentes de software construídos com diferentes tecnologias e com diversas origens;
- Capacidade da geração de componentes de software em tempo execução de acordo com as entradas do usuário e estado do servidor.

KAPPEL (2004) apresenta oito categorias de aplicações Web, mostradas na Figura 3. Essas categorias são apresentadas de acordo com sua evolução histórica

e níveis de complexidade. As categorias mais recentes apresentam maior complexidade.

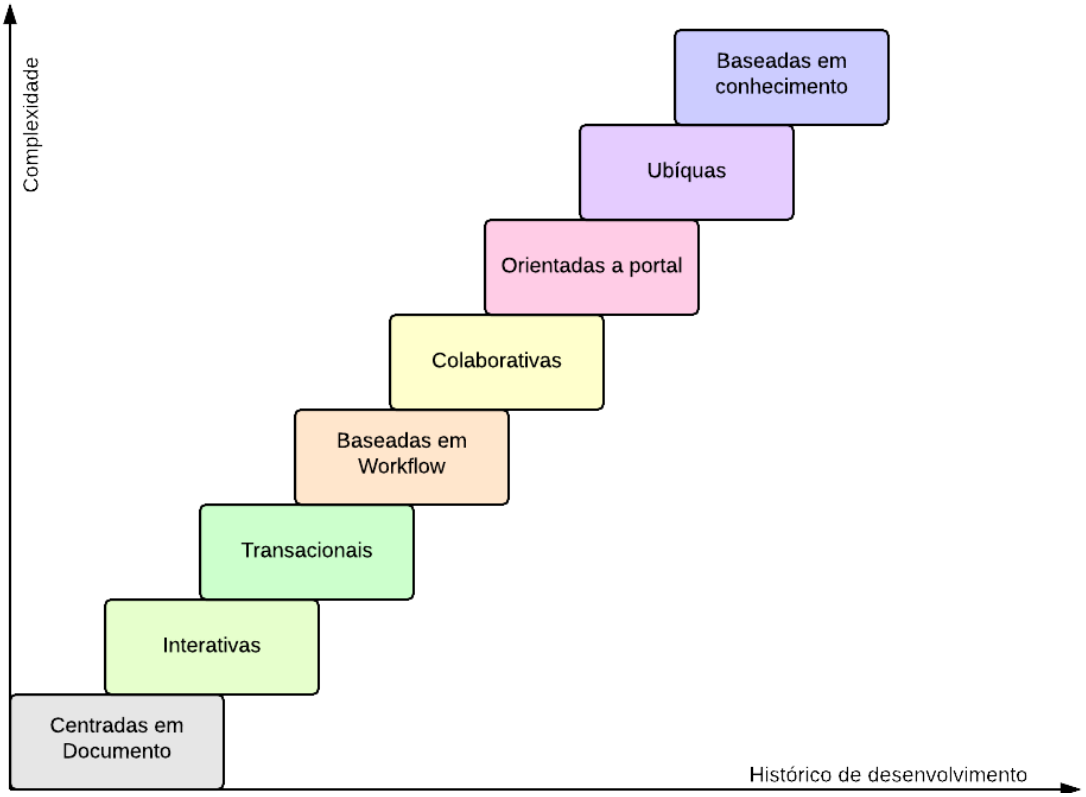

Figura 3: Categorias de Aplicações Web (ISABEL, 2011)

As categorias de aplicações Web são:

- Centradas em documentos: foram as primeiras aplicações Web, cujas páginas eram essencialmente estáticas;
- Interativas: aplicações que possibilitam uma pequena interação com o usuário, como por exemplo site de notícias e de planejamento de viagem;
- Transacionais: aplicações apoiadas por sistemas de banco de dados e que apresentam maior interatividade com o usuário, como *e-banking*;
- Baseada em *Workflow*: aplicações fundamentadas em um fluxo de trabalho, como aplicações para sistema de saúde;
- Aplicações Colaborativas: aplicações que viabilizam o trabalho cooperativo, como por exemplo, sala de bate-papo e *e-learning*;
- Orientadas a portal: aplicações que representam portais de comunidade, como compras *on-line*;

- Aplicações ubíquas: são aplicações que oferecem serviços dependentes do ambiente de utilização e independente de dispositivos;
- Baseadas em conhecimento: aplicações que aprendem com o uso, como Sistemas de recomendação.

2.2. Teste de Software

O teste de software é o processo de executar um programa ou sistema com a intenção de revelar falhas (MYERS, 1979). KANER (2006) complementa que o teste de software consiste em uma investigação experimental conduzida para prover informações aos usuários e envolvidos no processo sobre a qualidade do software testado.

O teste de software é um processo complexo que exige que sejam planejadas e executadas uma série de atividades. Para PRESSMAN (2006), uma estratégia de teste deve englobar planejamento, projeto de casos de teste, execução e avaliação de dados resultantes.

Testar um software com todas as entradas possíveis é em geral, algo impraticável, por isso é importante escolher entradas e ações que ofereçam maior probabilidade de acarretar um comportamento inadequado da aplicação (OLIVEIRA, 2007).

2.2.1. Teste de Interface Gráfica do Usuário

Atualmente, a maioria das aplicações de software fornece serviços aos seus usuários por meio de uma Interface Gráfica (do inglês, *Grafical User Interface* - GUI). Essa interface consiste de um conjunto de objetos virtuais (*widgets*) que visam tornar o software mais fácil de usar e é um dos principais componentes do software moderno (GANOV *et al*. 2009; RAUF *et al.*, 2010).

No teste GUI há uma interação direta com a aplicação simulando um usuário final. Esse tipo de teste contém uma série de eventos, tais como, campos de entradas e cliques de *mouse* que estão associados a cada interação do usuário com a aplicação.

MEMON *et al.* (2001) e MOREIRA e PAIVA (2014) apontam a quantidade de interações possíveis dos usuários como a grande dificuldade de testar GUI. DI LUCCA e FASOLINO (2006) destacam que a grande variedade de combinações

possíveis de todos os componentes envolvidos na execução de uma aplicação web torna inviável o teste de todos os cenários possíveis de utilização. O que normalmente ocorre é que apenas as combinações mais comuns são consideradas.

As ferramentas mais populares para teste GUI são as ferramentas de captura e reprodução (do inglês, *Capture-Replay*). Ao usar esse tipo de ferramenta, no modo de captura o testador interage com a aplicação sob teste (do inglês, *Aplication Under Test* - AUT). A ferramenta de captura armazena essa interação em um arquivo, que pode ser reproduzido posteriormente utilizando a ferramenta de reprodução (MEMON e SOFFA, 2003; XIE e MEMON, 2008).

A desvantagem de se utilizar ferramentas de captura e reprodução é que uma pequena modificação na GUI altera os casos de teste correspondentes, tornando cara a manutenção dos *scripts* de teste (AHO *et al*., 2011). MEMON *et al*. (1999) ressaltam que testadores que empregam essas ferramentas geralmente trabalham com uma pequena quantidade de casos de teste.

2.2.1.1. Caso de Teste GUI

Um caso de teste é composto por um conjunto de entradas, *script* de teste e saídas que direcionam o testador durante o teste (LEWIS, 2000; PRESMAN, 2006). As entradas de teste referem-se às condições iniciais, os *scripts* são os passos a serem executados, enquanto que as saídas de teste são os resultados esperados.

No teste GUI, os casos de teste possuem sequências de eventos como entrada e alguma indicação sobre o estado do programa (por exemplo, estado GUI e estado da memória) como resultado esperado (NGUYEN *et al.*, 2013).

2.2.1.2. Oráculo de Teste GUI

Oráculo de teste é um mecanismo que determina se um software foi executado corretamente em um caso de teste (XIE e MEMON, 2007). Em cada caso de teste GUI, projetistas de teste criam manualmente assertivas com os valores esperados para propriedades específicas de determinados *widgets*. Durante a execução do teste, essas assertivas são utilizadas como oráculos de teste para determinar se a GUI foi executada corretamente (XIE e MEMON, 2007).

2.2.2. Automação de Teste

Na automação do teste de software, tarefas de teste que seriam realizadas manualmente passam a ser executadas por um computador. Essas tarefas estão relacionadas tanto à geração quanto à execução dos casos de teste.

Embora a automação de teste de software seja vista como uma forma de melhorar a produtividade e a qualidade de software, é importante ressaltar que ela não é recomendada para todo tipo de teste. De acordo com FANTINATO (2004), a automação de teste é indicada para funcionalidades que envolvem a execução de tarefas repetitivas e cansativas, facilmente suscetíveis a enganos ou impossíveis de serem realizadas manualmente.

As principais técnicas de automação de teste são: captura e reprodução, programação de scripts, dirigida a dados e dirigida a palavra-chave (AHO *et al*., 2011; MARIANI *et al*., 2012).

2.2.3. Teste Baseado em Modelos

Teste Baseado em Modelos (TBM) é uma estratégia que envolve o desenvolvimento e utilização de um modelo que é essencialmente uma especificação de entradas para o software.

O objetivo do TBM é aumentar a eficiência e qualidade dos testes de software, automatizando processos de testes, de forma a diminuir o esforço de teste e evitar atividades sujeitas a enganos cometidos por seres humanos (SANTOS-NETO *et al*., 2007).

Os principais elementos de uma abordagem TBM são: o modelo do sistema, o algoritmo de geração de teste e as ferramentas que geram suporte para a infraestrutura dos testes (DALAL *et al.*, 1999). Na Figura 4 são mostradas as atividades do TBM.

Na abordagem TBM, o modelo estrutural ou comportamental construído a partir de artefatos de software permite a geração dos casos de teste, reduzindo assim o esforço gasto na geração dos mesmos.

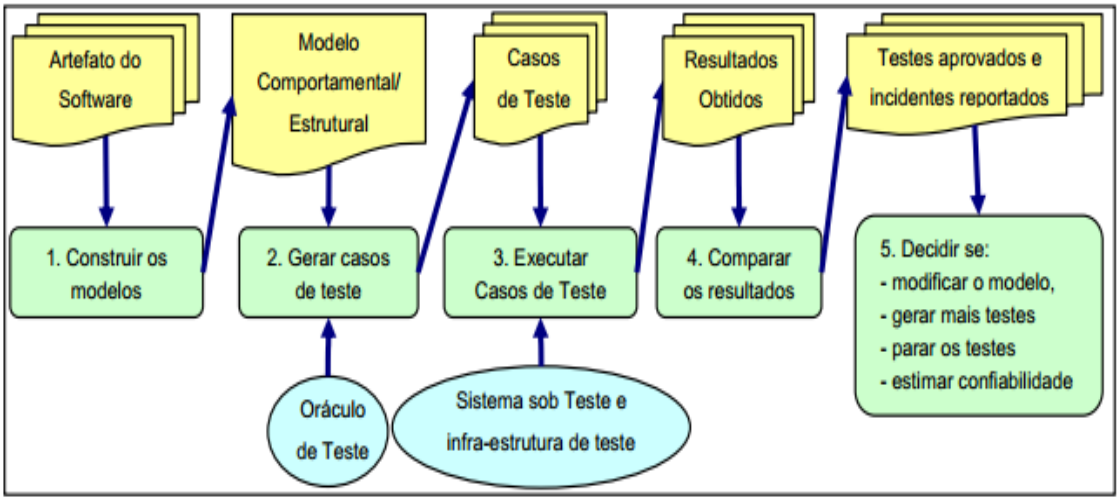

Figura 4: Atividades de TBM (DIAS-NETO, 2009)

A geração de casos de teste consiste na geração de dados (ou entradas) e geração das saídas esperadas. Após a geração dos casos de teste, os mesmos são executados para a AUT. Em seguida, os resultados obtidos são comparados com os resultados esperados. Quando há conformidade entre os resultados obtidos e os esperados, o caso de teste é aprovado; caso contrário, esses testes serão insumos para o relatório de incidentes. Por fim, é necessário decidir se o modelo será modificado, se serão gerados mais testes ou se é o momento de parar os testes, dentre outros.

As principais vantagens do TBM, segundo (DIAS-NETO e TRAVASSOS 2010), são:

- Redução de custo e esforço para planejamento/execução de teste;
- Melhoria da qualidade do produto final;
- Facilidade na comunicação entre as equipes de desenvolvimento e teste;
- Suporte a exposição de ambiguidade na especificação e projeto do software;
- Capacidade de gerar e executar automaticamente testes úteis e não reduntantes;
- Facilidade para atualizar um conjunto de casos de teste após realizar mudanças nos artefatos de software.

2.4. Trabalhos relacionados

MESBAH *et al.* (2012) propõem uma abordagem para analisar automaticamente as mudanças de estado na interface de Aplicações Web com tecnologia Ajax. A abordagem é baseada em um *crawler* que exercita o código do lado do cliente e identifica os elementos clicáveis que alteram o estado dentro do DOM (do inglês, *Dynamic Document Object*) construído dinamicamente no navegador. A partir dessas mudanças de estado, é inferido um gráfico de fluxo de estado que captura os estados GUI e as possíveis transições entre eles. Essa abordagem possui suporte de uma ferramenta de código aberto, denominada CRAWLJAX[1].

MARIANI *et al.* (2012) realizam testes em aplicações que interagem com os usuários por meio de uma GUI. O *AutoBlackTest* apresenta uma técnica para gerar automaticamente casos de teste em nível do sistema, utilizando reforço de aprendizagem para aprender a interagir com o aplicativo em teste e estimular suas funcionalidades.

AHO *et al.* (2013) apresentam uma abordagem e uma ferramenta, denominada *Murphy*, que permite extrair automaticamente modelos que podem ser usados no teste GUI. *Murphy* analisa a GUI ao interagir com a aplicação simulando um usuário final e gera uma Máquina de Estados Finitos (MEF) baseada no modelo comportamental observado durante a execução da aplicação. AHO *et al.* (2014) apresentam a avaliação da ferramenta *Murphy* em ambiente industrial. Essa avaliação demostra que os modelos extraídos podem ser utilizados para automatizar e apoiar diversas atividades de teste, incluindo suporte a testes manuais GUI, proporcionando redução no tempo de execução dos casos de teste existentes.

AMALFITANO *et al.* (2014) apresentam uma abordagem e uma ferramenta para análise dinâmica de aplicações Ricas (do inglês, *Rich Internet Applications* – RIAs) implementadas em Ajax, chamada *DynaRIA*. Essa ferramenta fornece um ambiente integrado para execução e análise da aplicação a partir de várias perspectivas, além de abstrair várias visualizações sobre a estrutura e o comportamento da aplicação em tempo de execução. Estudos envolvendo

[1] http://crawljax.com/

aplicações Ajax reais demostraram a utilidade e eficácia da ferramenta na compreensão, depuração, testes e cenários de avaliação da qualidade.

2.5. Web Guitar

A *Web Guitar* é uma extensão da ferramenta *Guitar* para aplicações Web. Segundo AHO *et al*. (2011), *Guitar* é uma das ferramentas mais avançadas de código aberto disponível para *download* e que permite criar automaticamente modelos estruturais para TBM com base na execução e engenharia reversa de uma aplicação existente.

Inicialmente, a ferramenta *Guitar* foi projetada para aplicações Java GUI. Porém, atualmente ela oferece suporte a outras plataformas, como Web e Android. As principais características da ferramenta *Guitar* são: automação das principais atividades de teste GUI, abordagem baseada em modelos e modularidade.

Web Guitar é composta pelos módulos apresentados na Figura 1, os quais são descritos a seguir:

1. *Web Ripper* é o módulo responsável por estender as funcionalidades do *Ripper* para aplicações Web;
2. Guitar:
    a) *Ripper* é o módulo responsável pela geração da árvore GUI (modelo estrutural GUI) por meio da execução da aplicação testada;
    b) Conversor de grafo é o módulo responsável pela conversão da árvore GUI em um grafo, denominado Grafo de Fluxo de Evento (do inglês, *Event Flow Graph* - EFG);
    c) Gerador de casos de teste é o módulo responsável pela geração automática de casos de teste;
    d) *Replayer* é o módulo responsável pela execução automática dos casos de teste gerados.
3. *Web Replayer* é o módulo responsável por estender as funcionalidades do *Replayer* para aplicações Web.

Enquanto o Conversor de grafo e o Gerador de casos de teste são independentes da plataforma, o *Ripper* e o *Replayer* interagem diretamente com a AUT, portanto exigem customização para a plataforma específica. Neste trabalho as

extensões dos módulos *Ripper* e *Replayer* que interagem com a aplicações Web serão tratados como *Web Ripper* e *Web Replayer*, respectivamente.

2.5.1. Modelo GUI

Uma GUI é modelada como um conjunto de objetos/*widgets* $O = \{o1, o2, ..., om\}$ e um conjunto de propriedades $P = \{p1, p2, ..., pl\}$ desses objetos.

O estado de uma GUI é o conjunto P de todas as propriedades de todos os objetos O que a GUI contém em um dado momento, na Figura 5 são mostrados dois estados de uma GUI, Estado 1 e Estado 2. Um conjunto válido de estados iniciais é associado a cada GUI. Um conjunto de estados SI é chamado de estados iniciais válidos para uma GUI particular se a GUI pode estar em qualquer estado $Si \in SI$ quando ele é invocado primeiro.

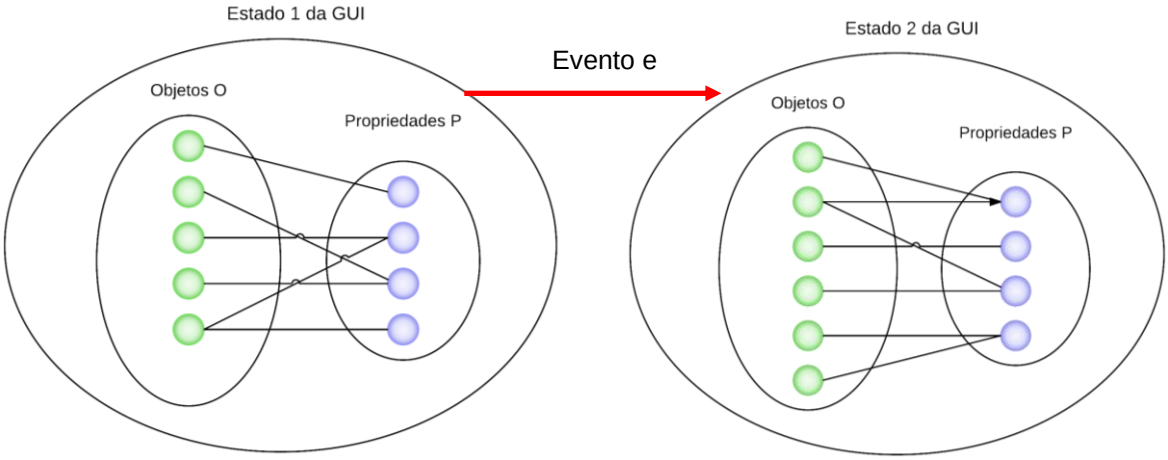

Figura 5: Modelo da GUI

O estado de uma GUI não é estático, pois eventos executados sobre a GUI alteraram seu estado. Esses estados são chamados estados alcançáveis.

Os eventos $E = \{e1, e2, ..., en\}$ associados a uma GUI são funções de um estado para outro estado da GUI, como mostrado na Figura 5, em que o Evento *e* altera uma GUI do Estado 1 para o Estado 2. Eventos ocorrem como parte de uma sequência de eventos. Uma sequência legal de eventos de uma GUI é $e1; e2; e3; ...; en,$ onde $ei+1$ pode ser executada imediatamente após ei.

Um caso de teste GUI T é um par $(S0, e1; e2; ...; en)$, que consiste de um estado $S0 \in SI$, chamado de estado inicial, e uma sequência legal de eventos $e1; e2; ...; en$.

Se o estado inicial de um caso de teste não é alcançado e/ou a sequência de eventos é ilegal, então o caso de teste é não executável.

2.5.2. Fluxo de Trabalho da Web Guitar

O fluxo de trabalho da ferramenta Web *Guitar* é apresentado na Figura 6. Primeiramente, o *Web Ripper* executa a aplicação automaticamente com o propósito de descobrir informações da GUI por meio de algoritmos automatizados e utilizando entradas manuais, as quais serão detalhadas na próxima seção.

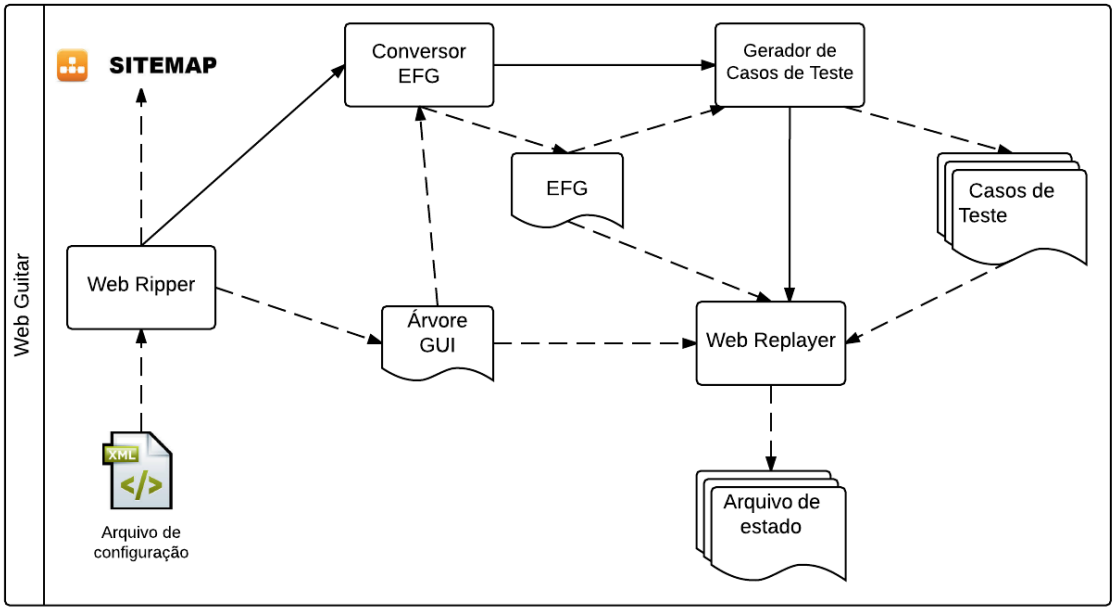

Figura 6: Fluxo de trabalho da ferramenta Web *Guitar* – (Adaptado de NGUYEN *et al.*, 2013)

Ao terminar de explorar a aplicação, o *Web Ripper* gera o modelo estrutural da mesma (árvore GUI) e também permite gerar o mapa do site, que mostra a navegação entre as páginas da aplicação.

O próximo módulo, o Conversor EFG utiliza a árvore GUI para gerar o Grafo de Fluxo de Evento (do inglês, *Event Flow Graph - EFG*) que possui todos os eventos identificados e as interações entre os mesmos.

O EFG é um arquivo *XML* que serve como insumo para que o módulo Gerador de Casos de teste gere de fato os casos de teste, compostos pela sequência de eventos desejada. Durante à geração de casos de teste, há uma redução dos casos de teste duplicados e/ou não alcançáveis.

O *Web Replayer* reúne informações da árvore GUI, do EFG e de um caso de teste para executar o teste definido partindo do mesmo estado inicial utilizado pelo

Web Ripper. Para cada evento no caso de teste, o *Web Replayer* usa as informações contidas na árvore GUI e no EFG para identificar as páginas e os *componentes* que o evento precisa para ser executado. Ao final deste processo é gerado um arquivo de estado (do inglês, *State File* - STA) que contêm o estado da aplicação Web após cada etapa intermediária do caso de teste, em que o estado da aplicação é a coleção de páginas atuais e seus componentes correspondentes.

É importante ressaltar que o *Web Replayer* oferece suporte a dois oráculos de teste. O primeiro chamado Verificador de Falhas (do inglês, *Crash Verifier)* que relata falhas na execução do caso de teste. O segundo chamado verificador de Estado que combina estados de saída GUI em diferentes execuções do caso de teste.

2.5.3. Conceitos relacionados a execução da Web Guitar

Nesta seção são descritos conceitos utilizados na execução da *Web Guitar*.

2.5.3.1. Entradas manuais

Originalmente na ferramenta *Guitar*, as entradas manuais são constituídas por componentes ignorados, dados de entrada e componentes terminais. Os primeiros referem-se aos componentes que não se deseja explorar ou que demandam muito tempo para serem explorados. Os dados de entrada se referem aos valores para os campos de texto. Já os componentes terminais são botões que submetem informações de formulários e/ou fecham páginas. Na ferramenta *Web Guitar*, os componentes terminais são definidos como os botões que geralmente estão no final da página, como por exemplo, os botões *Save* e *Cancel* mostrados na Figura 7.

2.5.3.2. Largura e Profundidade

A largura refere-se ao número máximo de links que podem ser explorados em uma única página. Já a profundidade refere-se à quantidade máxima de níveis que podem são alcançados a partir da URL informada. Na Figura 8 é mostrada a página index da aplicação Web MTCONTROOL. Partindo-se dessa página e utilizando-se profundidade dois é possível alcançar as páginas cujos links estão destacados. Para alcançar a página *Login* seria necessário utilizar-se no mínimo a largura quatro.

Figura 7: Componentes terminais

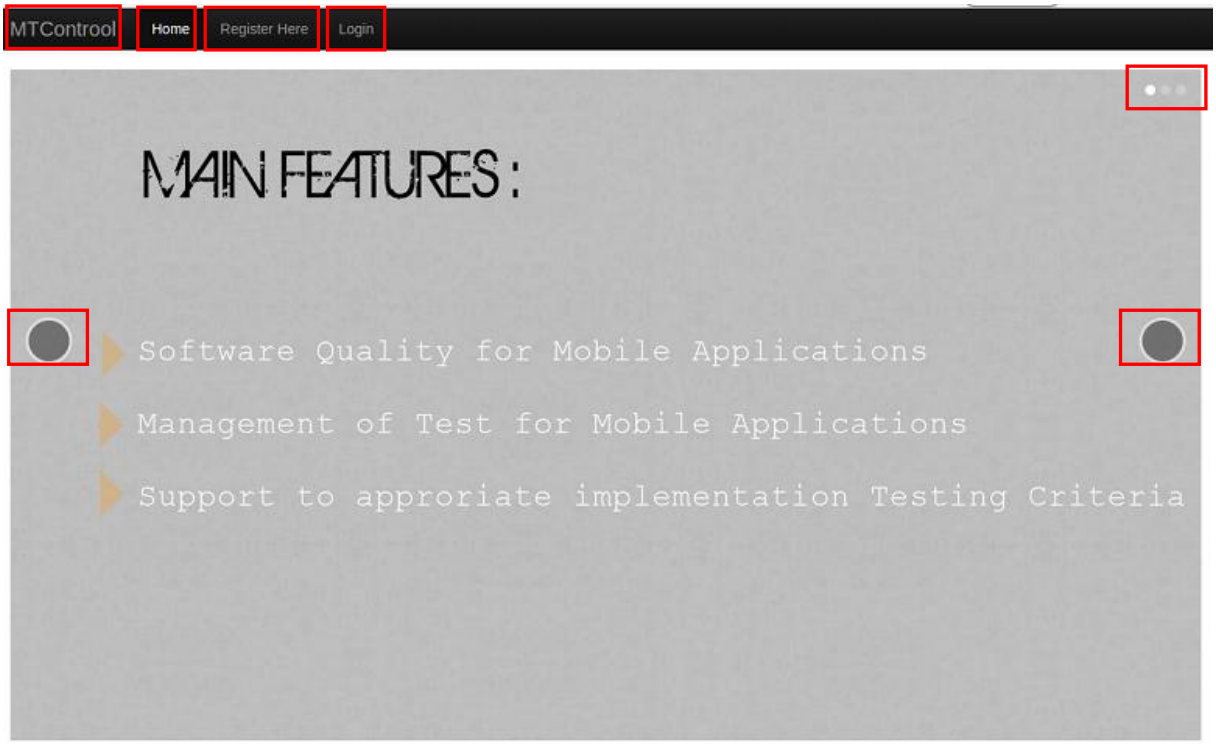

Figura 8: Página index do MTCONTROOL

2.5.3.3. Árvore GUI e Grafo EFG

A árvore GUI refere-se ao modelo estrutural da aplicação que contém o estado de todas as páginas executadas pelo *Web Ripper.* Trata-se de um arquivo no formato XML.

O grafo EFG é um grafo em que cada nó representa um evento, e todos os eventos que podem ser executados imediatamente após um determinado evento possuem uma aresta partindo do mesmo.

2.5.4. Execução da Web Guitar

Para exemplificar a execução da *Web Guitar* será utilizado o sistema *Simple Student Information System*, denominado SIS3. Esse sistema gerencia informações de estudantes e permite criar, editar e excluir um estudante e ainda permite listar todos os estudantes cadastrados. A página *index* do SIS3 é mostrada na Figura 9.

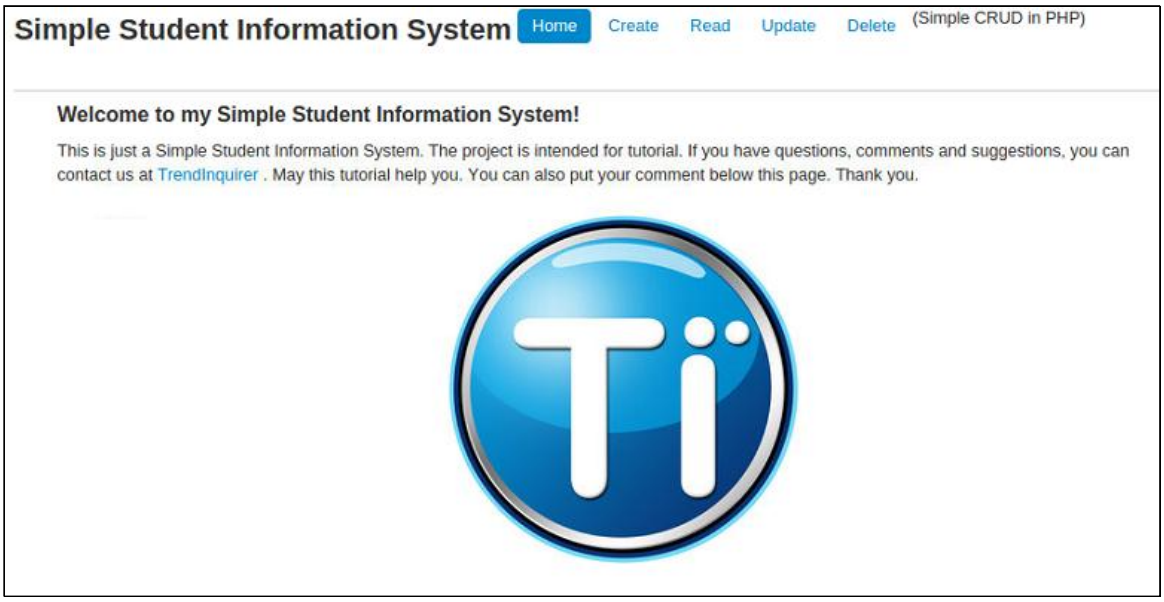

Figura 9: Página index do SIS3

Os módulos do *Web Ripper* são executados por meio da linha de comandos do Linux. Para executar o *Web Ripper*, deve-se digitar o comando indicado na Figura 10, onde:

```
silvia@silvia: ~/Documentos/guitar/dist/guitar
silvia@silvia:~/Documentos/guitar/dist/guitar$ ./sel-ripper.sh -u http://localhost/SIS3
-d 2 -w 3 -c configurationFile.xml -g SIS3.GUI
```

Figura 10: Comando para executar o *Web Ripper*

- -u: URL a partir da qual a aplicação será explorada;
- -d: Profundidade;
- -w: Largura;
- -c: Arquivo de configuração;
- -g: Arquivo referente a árvore GUI;
- -gf: Mapa do site (imagem e arquivo .DOT);

Caso não sejam definidos valores para largura, profundidade, árvore GUI e arquivo de configuração, eles assumem os valores, 1, 3, *GUITAR-Default.GUI* e *configuration.xml*, respectivamente.

Assim que o *Web Ripper* é iniciado, é aberta a página mostrada na Figura 11, a partir da qual ele é executado.

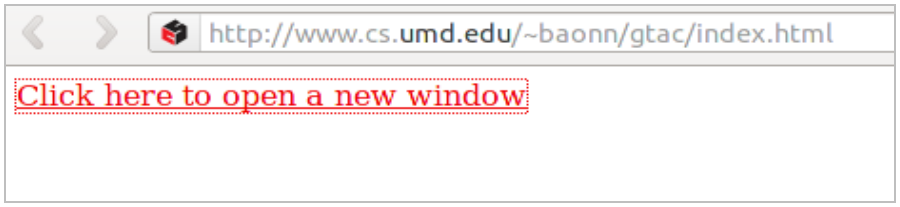

Figura 11: Página inicial para executar o *Web Ripper*

A saída do *Web Ripper* é o arquivo de árvore GUI, mostrado na Figura 12.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GUIStructure>
    <GUI>
        <Window>
            <Attributes>
                <Property>
                    <Name>Title</Name>
                    <Value>http://localhost/SIS3/index.php</Value>
                </Property>
                <Property>
                    <Name>Modal</Name>
                    <Value>true</Value>
                </Property>
                <Property>
                    <Name>Rootwindow</Name>
                    <Value>false</Value>
                </Property>
            </Attributes>
        </Window>
        <Container>
            <Contents>
                <Container>
                    <Attributes>
                        <Property>
                            <Name>ID</Name>
                            <Value>w1650443748</Value>
                        </Property>
```

Figura 12: Parte da árvore GUI referente a página index do SIS3

Para executar o Conversor de grafo, deve-se digitar o comando da Figura 13, onde:

```
silvia@silvia: ~/Documentos/guitar/dist/guitar
silvia@silvia:~/Documentos/guitar/dist/guitar$ ./gui2efg.sh -g SIS3.GUI -e SIS3.EFG
```

Figura 13: Comando para executar o Conversor de grafo

- -g: Arquivo da árvore GUI;
- -e: Arquivo EFG;

Caso não seja informado um nome para o arquivo EFG, ele assumirá o nome *GUITAR-Default.EFG*. A saída do Conversor é o grafo EFG mostrado na Figura 14.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<EFG>
    <Events>
        <Event>
            <EventId>e2408267416</EventId>
            <WidgetId>w2408267416</WidgetId>
            <Type>RESTRICED FOCUS</Type>
            <Initial>true</Initial>
            <Action>edu.umd.cs.guitar.event.WebEvent</Action>
            <Name></Name>
        </Event>
        <Event>
            <EventId>e3887804956</EventId>
            <WidgetId>w3887804956</WidgetId>
            <Type>RESTRICED FOCUS</Type>
            <Initial>true</Initial>
            <Action>edu.umd.cs.guitar.event.WebEvent</Action>
            <Name></Name>
        </Event>
        <Event>
            <EventId>e136859784</EventId>
            <WidgetId>w136859784</WidgetId>
            <Type>RESTRICED FOCUS</Type>
            <Initial>true</Initial>
            <Action>edu.umd.cs.guitar.event.WebEvent</Action>
            <Name></Name>
        </Event>
        <Event>
            <EventId>e2573629528</EventId>
            <WidgetId>w2573629528</WidgetId>
            <Type>SYSTEM INTERACTION</Type>
            <Initial>true</Initial>
            <Action>edu.umd.cs.guitar.event.WebEvent</Action>
            <Name></Name>
        </Event>
```

Figura 14: Parte do arquivo EFG

Para executar o gerador dos casos de teste, deve-se digitar o comando da Figura 15, onde:

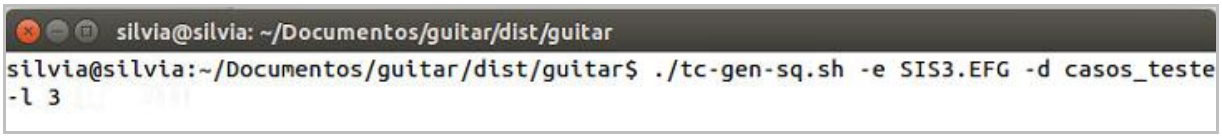

Figura 15: Comando para executar o Gerador de casos de teste

- -e: Arquivo EFG;
- -l: Tamanho dos casos de testes desejados, ou seja, a quantidade de eventos que serão executados.
- -d: Diretório onde serão gerados os casos de teste.
- -m: é o número máximo de casos de teste que serão gerados.

Na Figura 16, são mostrados alguns casos de teste gerados.

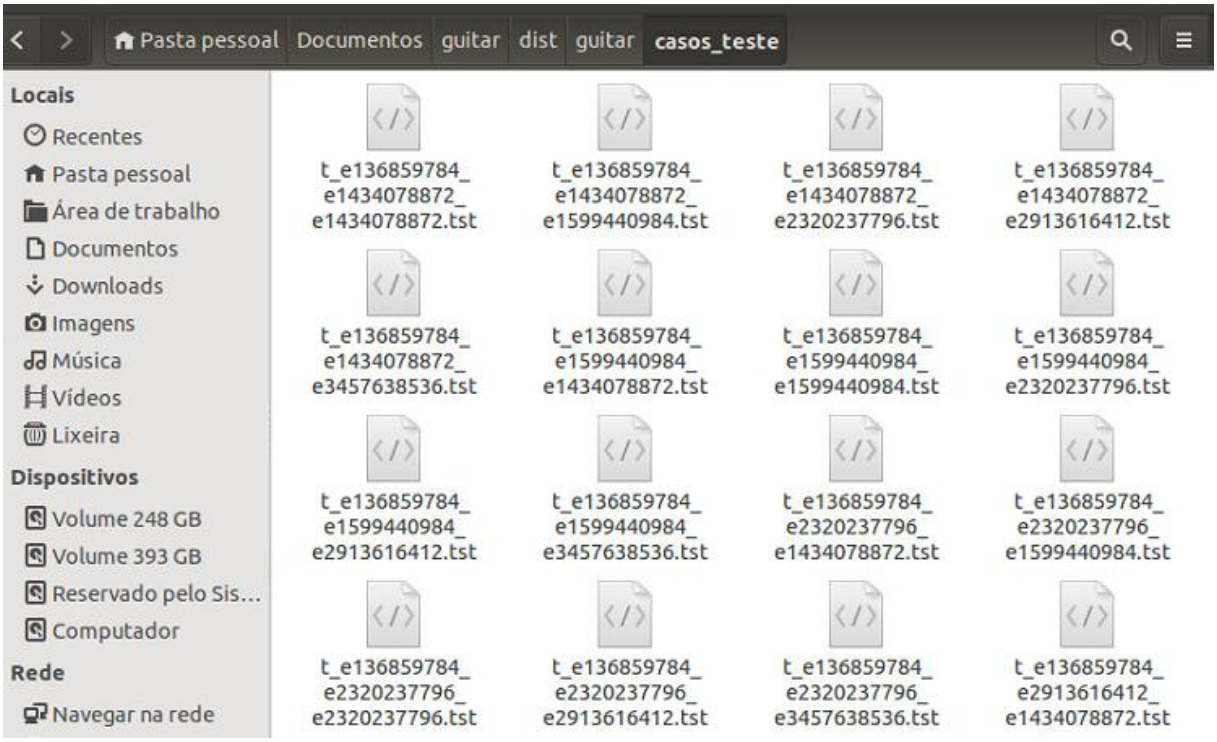

Figura 16: Casos de teste gerados

Para executar o *Web Replayer*, deve-se digitar o comando da Figura 17, onde:

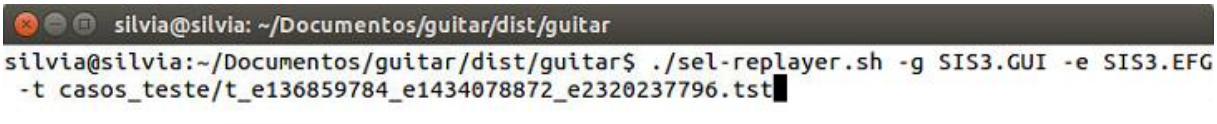

Figura 17: Comando para executar o Web Replayer

- -g: Árvore GUI gerada pelo *Web Ripper*;
- -e: Arquivo EFG gerada pelo conversor EFG;

- -t: Nome do caso de teste que será executado.

A saída do Conversor do *Web Replayer* é o arquivo de estado apresentado na Figura 18.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TestCase>
    <GUIStructure/>
    <Step>
        <EventId>e136859784</EventId>
        <ReachingStep>false</ReachingStep>
        <GUIStructure>
            <GUI>
                <Window>
                    <Attributes>
                        <Property>
                            <Name>Title</Name>
                            <Value>http://localhost/SIS3/read.php</Value>
                        </Property>
                        <Property>
                            <Name>Modal</Name>
                            <Value>true</Value>
                        </Property>
                        <Property>
                            <Name>Rootwindow</Name>
                            <Value>false</Value>
                        </Property>
                    </Attributes>
                </Window>
                <Container>
                    <Contents>
                        <Container>
                            <Attributes>
<Property>
    <Name>ID</Name>
    <Value>w3655382268</Value>
</Property>
<Property>
    <Name>Class</Name>
    <Value>org.openqa.selenium.firefox.FirefoxWebElement</Value>
</Property>
```

Figura 18: Parte do arquivo de estado gerado

2.6. Estrutura Web Guitar

Nesta seção é apresentada a estrutura da ferramenta *Web Guitar*, com suas principais interfaces e classes. Na Figura 19 é apresentado o Diagrama de Classes dessa ferramenta. Esse diagrama contém as principais classes utilizadas neste trabalho, as quais estão divididas em quatro pacotes. Os pacotes *gui-model-core* e *gui-ripper-core* contém as classes utilizadas pelo módulo *Ripper*, enquanto os pacotes *gui-model-sel* e *gui-ripper-sel* contém as classes que mapeiam os objetos do

módulo *Ripper* em objetos Web, ou seja, as classes utilizadas pelo módulo *Web Ripper*.

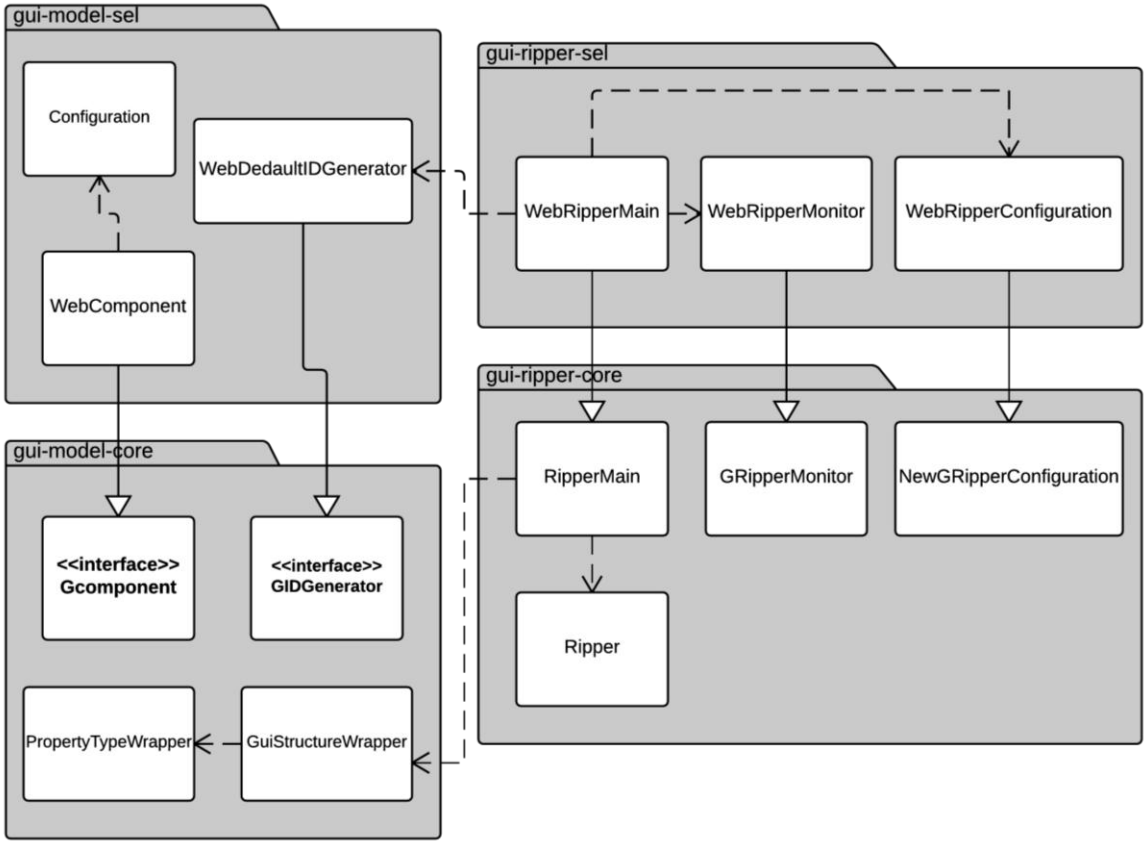

Figura 19: Diagrama de Classes Web Guitar

As principais interfaces utilizadas pela *Guitar* são:

- *GIDGenerator:* Gera identificadores (IDs) para cada componente GUI;
- *GComponent*: Representa um componente GUI e métodos para acessar suas propriedades;
- *GEvent*: Representa um tipo de evento, como por exemplo, entrada de texto. Um *GEvent* associado a um *GComponent* representa um evento de uma GUI específica.

Para estender os recursos da *Guitar* para o domínio de Aplicações Web, as interfaces devem ser implementadas por classes da *Web Guitar.*

As interfaces *GIDGenerator*, *GComponent* e *GEvent* são implementadas pelas classes WebDefaultIDGenerator, *WebComponent* e *WebEvent* respectivamente.

Além das interfaces citadas anteriormente, a ferramenta *Guitar* possui as seguintes classes:

- *NewGRipperConfiguration:* Recebe parâmetros informados por meio da linha de comando como largura e profundidade mostrados na Figura 10.
- *RipperMain:* Controla a execução do *Ripper;*
- *Ripper*: Responsável por explorar da aplicação sob teste.
- *GRipperMonitor*: Monitora informações das GUIs que serão exploradas pelo *Ripper*;
- *Configuration*: Manipula os parâmetros do arquivo de configuração, componentes terminais e páginas ignoradas;
- GuiStructureWrapper: Manipula a árvore GUI;
- PropertyTypeWrapper: Manipula as propriedades da árvore GUI.

As classes *NewGRipperConfiguration*, *RipperMain* e *GRipperMonitor* são estendidas pelas classes *WebRipperConfiguration*, *WebRipperMain* e *WebRipperMonitor*, respectivamente.

A classe *WebRipperConfiguration* inclui parâmetros específicos da *Web* como URL e Mapa do site. Já a classe *WebRipperMain é responsável por inicializar as classes WebRipperMonitor e WebDefaultIDGenerator.*

2.7. Mapeamento Sistemático sobre Técnicas de TBM para Aplicações Web

Com o propósito de se obter conhecimento sobre as abordagens de TBM para aplicações Web descritas na literatura técnica, foi planejado e executado um Mapeamento Sistemático. Estudos de Mapeamento Sistemático (MS) são projetados para fornecer uma visão geral de uma área de investigação para determinar se existe evidência de pesquisa sobre um tema e fornecer uma indicação da quantidade de provas. Os resultados de um MS pode identificar áreas adequadas para a realização de uma Revisão Sistemática da Literatura e também áreas nas quais um estudo preliminar é mais apropriado (KITCHENHAM e CHARTERS 2007).

2.7.1. Definição do Objetivo e Questões de Pesquisa

O MS foi realizado no período de Novembro/2013 à Março/2014 e seu objetivo foi estruturado segundo o paradigma GQM (*Goal/Question/Metric*) (BASILI *et al.*, 1994), o qual é apresentado na Tabela 1.

Tabela 1: Objetivo segundo o paradigma GQM

Analisar	Técnicas de Teste Baseado em Modelos em Aplicações Web
Com o propósito de	Caracterizar
Com relação a	Modelos adotados para representação do software
Do ponto de vista do	Pesquisador
No contexto	Pesquisas acadêmicas em engenharia de software

Baseado no objetivo da pesquisa foram formuladas três questões de pesquisa principais, as quais são apresentadas a seguir:

- Q1 – Quais os modelos utilizados para mapear o comportamento de aplicações web relacionados à geração automática de casos de teste?
  - Q1.1 – Quais desses são modelos que representam a interface do sistema?
  - Q1.2 – Quais desses são modelos descritos na UML 2.X?
- Q2 – Que características de qualidade específicas para aplicações Web tais modelos conseguem representar e prover testes?
- Q3 – Que tipos de teste tais técnicas conseguem aplicar em um software?
- Q4 – Qual a localização dos testes?

2.7.2. Fontes de Pesquisa e String de busca

Para atingir o objetivo da proposta e responder as questões de pesquisa, a string de busca foi estruturada de acordo com a estratégia PICO (*Population*, *Intervention*, *Comparison* e *Outcomes*) e definidas da seguinte forma:

(P) População: Aplicações web.

(I) Intervenção: Teste baseado em Modelos.

(C) Comparação: Não se aplica.

(O) Resultados: Técnica, abordagem, método, metodologia, ferramenta, processo e estratégia.

O resultado final é:

- **P =** ("web development" OR "web software" OR "web application" OR "web applications" OR "web system" OR "web systems" OR "web engineering" OR "internet engineering" OR "website" OR "websites" OR "web site" OR "web sites" OR "web based-system" OR "web based-systems" OR "web based-application" OR "web based-applications" OR "e-commerce"

OR "networking system" OR "network system" OR "networked system" OR "web gui" OR "web guis")

AND

- **I =** ("specification based test" OR "specification based testing" OR "specification driven test" OR "specification driven testing" OR "interface based test" OR "interface based testing" OR "interface driven test" OR "interface driven testing" OR "gui based test" OR "gui based testing" OR "gui driven test" OR "gui driven testing" OR "gui test" OR "gui testing" OR "model based test" OR "model based testing" OR "model driven test" OR "model driven testing" OR "mbt" OR "uml based test" OR "uml based testing" OR "uml driven test" OR "uml driven testing" OR "use case driven test" OR "use case driven testing" OR "requirement based test" OR "requirement based testing" OR "requirement driven test" OR "requirement driven testing" OR "scenario based test" OR "scenario based testing" OR "early test" OR "early testing")

AND

- **O =** ("technique" OR "approach" OR "method" OR "procedure" OR "methodology" OR "tool" OR "process" OR "strategy")

Como pode ser observado acima, os termos referentes a população e a intervenção apresentaram variações de singular e plural, porém os termos referentes aos resultados não apresentaram essas variações durante o ajuste da String, que foi realizado para cada biblioteca digital identificada a seguir.

As fontes de pesquisa para obtenção dos resultados utilizadas foram bibliotecas digitais e fontes manuais, conforme listadas abaixo:

1. Bibliotecas digitais:
   - Scopus: <http://www.scopus.com>;
   - IEEE Computer Science Digital Library: <http://ieeexplore.ieee.org>.
2. Fontes Manuais:
   - SBES: Simpósio Brasileiro de Engenharia de Software;
   - SBQS: Simpósio Brasileiro de Qualidade de Software;
   - SAST: Brazilian Workshop on Systematic and Automated Software Testing.

2.7.3. Critérios de inclusão dos artigos

Foram utilizados os critérios de inclusão listados a seguir:

1. As publicações devem estar em Português ou Inglês;

2. As publicações devem estar disponíveis na Web ou por meio do contato com os autores;
3. As publicações devem descrever a geração automática de casos de teste para sistemas web*;*
4. As publicações devem mencionar os modelos utilizados na geração de casos de teste.

Qualquer estudo que não satisfaça a todos os critérios citados deverá ser excluído.

2.7.4. Formulário de Extração de Dados

Inicialmente, foram identificadas 159 publicações. Após a aplicação dos critérios de inclusão, esse número foi reduzido para 57 artigos dos quais foram extraídos os dados da lista apresentada na Tabela 2.

Tabela 2: Campos extraídos no mapeamento sistemático

Campo	Descrição
Título	Título do trabalho.
Autores	Autores do trabalho.
Ano de publicação	Ano em que o trabalho foi publicado.
Local de publicação	Local (conferência/periódico) onde o trabalho foi publicado.
Tipo de contribuição	Tipo de contribuição produzida pelo trabalho. As categorias são: Modelo, Técnica, Ferramenta e Processo.
Tipo de estudo	Tipo de artigo, ao qual se refere o trabalho. Essas categorias são as mesmas utilizadas em Garousi *et al.* (2013): • Solução Proposta: Uma solução para um problema é proposta, onde os potenciais benefícios e aplicabilidade da solução são mostrados; • Prova de conceito: Técnicas investigados são novas e ainda não foram implementadas na prática. Técnicas utilizadas são, por exemplo, experimentos, ou seja, o trabalho é realizado no laboratório; • Avaliação experimental: Técnicas são implementados na prática e uma avaliação da técnica é conduzida. Mostra-se como a técnica é implementada na prática e as consequências de sua implementação; • Relatos de Experiência: Explicam como algo tem sido feito na prática, refere-se a experiência pessoal do(s) autor(es).
Localização dos Testes	Ambiente computacional de realização dos testes em aplicações web. As categorias são: Servidor, Cliente ou Ambos.
Modelo	Modelo que descreve o comportamento ou estrutura da aplicação sob teste. As

utilizado	categorias são: Baseado em UML, Máquina de Estados, *Workflow*, Rede de Petri, Cadeias de *Markov* ou outros.
Nível de teste	Referem-se aos níveis de teste avaliados no trabalho proposto. As categorias são: Unidade, Integração, Sistema, Aceitação e Regressão.
Características de qualidade	Refere-se a características de Qualidade (NBR ISO/IEC 9126-1, 2003) avaliadas no trabalho proposto. Confiabilidade, Manutenibilidade, Usabilidade, Portabilidade, Eficiência e Funcionalidade. Neste trabalho será adotada a categorização utilizada por Dias-Neto (2009), na qual a subcategoria Segurança (relacionada ao grupo Funcionalidade) foi classificada separadamente, como sendo uma característica de qualidade. Além disso, a categoria Confiabilidade foi incorporada à categoria Funcionalidade.
Existência de apoio ferramental	Indica se o trabalho descreve (ou não) a geração de alguma ferramenta de apoio.

Alguns campos permitem mais de um valor. Para estes, a soma das categorias analisadas pode ser superior a 57, que representa a quantidade de artigos analisados.

A Tabela 3 apresenta uma listagem com as principais características dos trabalhos retornados no mapeamento sistemático.

Tabela 3: Caracterização dos Trabalhos Selecionados no Mapeamento Sistemático.

Autores	Título	Local	Ano	Contribuição	Tipo Artigo	Local Testes	Modelos	Característica de Qualidade
Anisetti *et al.*	A test-based security certification scheme for Web services	ACM Transactions on the Web	2013	Outros	Prova de conceito	Servidor	Máquina de Estados	Segurança
Arora e Sinha	A sustainable approach to automate user session based Máquina de Estados generation for AJAX web applications	Journal of Theoretical and Applied Information Technology	2013	Técnica e Ferramenta	Avaliação experimental	Cliente/ Servidor	Máquina de Estados	Não definido
Athira e Samue	Web services regression test case prioritization	International Conference on Computer Information Systems and Industrial Management Applications	2010	Técnica	Proposta de solução	Servidor	Baseado em UML	Manutenibilidade
Bansal e Sabharwal	A model based approach to test case generation for testing the navigation behavior of dynamic web applications	International Conference on Contemporary Computing	2013	Técnica	Proposta de solução	Cliente/ Servidor	Workflow	Funcionalidade e Usabilidade
Belli e Linschulte	Event-driven modeling and testing of real-time web services	Service Oriented Computing and Applications	2010	Técnica	Prova de conceito	Cliente/ Servidor	Workflow	Funcionalidade e Eficiência
Belli *et al.*	Event-based GUI testing and reliability assessment techniques An experimental insight and preliminary results	International Conference on Software Testing, Verification, and Validation	2011	Outros	Avaliação experimental	Não definido	Workflow	Funcionalidade
Belli *et al.*	Model-based testing of web service compositions	International Symposium on Service-Oriented System Engineering	2011	Técnica	Proposta de solução	Servidor	Workflow	Funcionalidade
Belli *et al.*	A holistic approach to model-based testing of Web service compositions	Software – Practice and Experience	2012	Técnica	Prova de conceito	Servidor	Workflow	Não definido
Bertolini e Mota	A Framework for GUI Testing based on Use Case Design	International Conference on Software Testing, Verification, and Validation Workshops	2010	Processo	Avaliação experimental	Servidor	Outros	Funcionalidade e Usabilidade
Bolis *et al.*	Model-driven testing for web applications using abstract State machines	Lecture Notes in Computer Science	2012	Técnica	Prova de conceito	Não definido	Máquina de Estados	Funcionalidade
Boumiza e Azzouz	Design and development of a user interface to customize web testing scenarios	International Conference on Education and e-Learning Innovations	2012	Técnica	Proposta de solução	Não definido	Outros	Não definido
Bozic e Wotawa	XSS pattern for attack modeling in testing	International Workshop on Automation of Software Test	2013	Técnica	Proposta de solução	Servidor	Baseado em UML	Segurança
Bryce *et al.*	Developing a single model and test prioritization strategies for event-driven software	IEEE Transactions on Software Engineering	2011	Modelo	Avaliação experimental	Servidor	Workflow	Não definido
Casado *et al.*	Testing the reliability of web services transactions in cooperative applications	ACM Symposium on Applied Computing	2012	Técnica e Ferramenta	Prova de conceito	Servidor	Baseado em UML	Funcionalidade
Copstein e Oliveira	Automated Test Script Generation for Model-Based Testing	Brazilian Symposium on Software Quality	2005	Técnica	Prova de conceito	Servidor	Cadeia de Markov	Não definido

Autores	Título	Local	Ano	Contribuição	Tipo Artigo	Local Testes	Modelos	Característica de Qualidade
Da Costa *et al.*	RSA-MBT: A test tool for generating test artifacts based on models	European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering	2013	Ferramenta	Proposta de solução	Não definido	Baseado em UML	Não definido
Escott *et al.*	Integrating model-based testing in model-driven web engineering	Asia-Pacific Software Engineering Conference	2011	Técnica	Prova de conceito	Servidor	Baseado em UML	Funcionalidade
Frajtak *et al.*	Formal specification to support advanced model based testing	Federated Conference on Computer Science and Information Systems	2012	Modelo	Proposta de solução	Cliente/ Servidor	Máquina de Estados	Usabilidade
Frajták *et al.*	Manual testing of web software systems supported by direct guidance of the tester based on design model	World Academy of Science, Engineering and Technology	2011	Técnica	Proposta de solução	Servidor	Máquina de Estados	Manutenibilidade
Francisco *et al.*	Turning web services descriptions into QuickCheck models for automatic testing	Proceedings of the ACM SIGPLAN International Conference on Functional Programming	2013	Técnica	Proposta de solução	Servidor	Outros	Funcionalidade
George *et al.*	Specification for testing	Lecture Notes in Computer Science	2007	Técnica	Proposta de solução	Cliente	Outros	Segurança e Eficiência
Hajiabadi e Kahani	An automated model based approach to test web application using ontology	IEEE Conference on Open Systems	2011	Técnica	Prova de conceito	Cliente/ Servidor	Outros	Funcionalidade
Hauptmann e Junker	Utilizing user interface models for automated instantiation and execution of system tests	International Workshop on End-to-End Test Script Engineering	2011	Técnica	Prova de conceito	Cliente	Outros	Manutenibilidade
Hossen *et al.*	Automatic generation of test drivers for model inference of web applications	International Conference on Software Testing, Verification and Validation	2013	Técnica e Ferramenta	Prova de conceito	Cliente/ Servidor	Máquina de Estados	Segurança
Keum e Kang	Scenario-based testing of converged applications	International Conference on Computing and Convergence Technology	2012	Técnica	Proposta de solução	Cliente	Baseado em UML	Funcionalidade
Koopman *et al.*	Model-based testing of thin-client web applications and navigation input	Lecture Notes in Computer Science	2007	Técnica	Proposta de solução	Servidor	Máquina de Estados	Usabilidade
Koopman *et al.*	Model-based testing of thin-client web applications	Lecture Notes in Computer Science	2006	Técnica	Proposta de solução	Cliente	Máquina de Estados	Portabilidade
Kuemper *et al.*	Test derivation for semantically described IoT services	Future Network and Mobile Summit	2013	Técnica	Proposta de solução	Cliente	Outros	Não definido
Lebeau *et al.*	Model-based vulnerability testing for web applications	International Conference on Software Testing, Verification and Validation Workshops	2013	Técnica	Proposta de solução	Cliente/ Servidor	Baseado em UML	Segurança

Autores	Título	Local	Ano	Contribuição	Tipo Artigo	Local Testes	Modelos	Característica de Qualidade
Lemos *et al.*	Visualization, Analysis, and Testing of Java and AspectJ programs with multi-level system graph	Brazilian Symposium on Software Engineering	2013	Técnica e Ferramenta	Prova de conceito	Servidor	Outros	Não definido
Lenz *et al.*	Model driven testing of SOA-based software	CEUR Workshop Proceedings	2007	Outros	Proposta de solução	Servidor	Baseado em UML	Portabilidade
Li *et al.*	A framework of model-driven web application testing	International Computer Software and Applications Conference	2006	Técnica	Proposta de solução	Cliente/ Servidor	Baseado em UML	Não definido
Li *et al.*	Web application model recovery for user input validation testing	International Conference on Software Engineering Advances	2007	Técnica e Ferramenta	Avaliação experimental	Servidor	Baseado em UML	Usabilidade
Oliveira *et al.*	Performance Testing from UML Models with Resource Descriptions	Brazilian Workshop on Systematic and Automated Software Testing	2007	Técnica e Ferramenta	Proposta de solução	Cliente	Redes de Petri	Eficiência
Perez *et al.*	Automação em Projeto de Testes Usando Modelos UML	Brazilian Workshop on Systematic and Automated Software Testing	2007	Técnica e Ferramenta	Relato de Experiência	Cliente	Baseado em UML	Funcionalidade
Pinheiro *et al.*	Model-Based Testing of RESTful Web Services Using UML Protocol State Machines	Brazilian Workshop on Systematic and Automated Software Testing	2013	Técnica e Ferramenta	Proposta de solução	Servidor	Baseado em UML	Funcionalidade
Pinto e Staa	Functional Validation Driven by Automated Tests	Brazilian Symposium on Software Engineering	2013	Processo	Proposta de solução	Cliente	Outros	Funcionalidade
Qian e Chen	Generating test case specifications of web service composition using model checking	Journal of Shanghai University	2011	Técnica	Proposta de solução	Cliente	Máquina de Estados	Funcionalidade
Reza *et al.*	A model based testing technique to test web applications using StateCharts	International Conference on Information Technology: New Generations	2008	Técnica	Proposta de solução	Cliente/ Servidor	Baseado em UML	Funcionalidade e Usabilidade
Schur *et al.*	Mining behavior models from enterprise web applications	European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software	2013	Técnca e Ferramenta	Evaluation reserach	Cliente/ Servidor	Máquina de Estados	Manutenibilidade
Shahzad *et al.*	Automated optimum test case generation using web navigation graphs	International Conference on Emerging Technologies	2009	Técnica	Proposta de solução	Servidor	Workflow	Funcionalidade
Silva *et al.*	Testes de Aceitação: Uma Abordagem Baseada na Especificação de Casos de Uso e no Design da Interação	Brazilian Workshop on Systematic and Automated Software Testing	2011	Técnica	Prova de conceito	Cliente/ Servidor	Outros	Usabilidade
Souza *et al.*	Testes Funcionais de Web Services RESTful a partir de Modelos UML	Brazilian Symposium on Software Quality	2012	Técnica	Prova de conceito	Servidor	Outros	Portabilidade
Sprenkle *et al.*	A study of usage-based navigation models and generated abstract test cases for web applications	International Conference on Software Testing, Verification, and Validation	2011	Outros	Avaliação experimental	Servidor	Cadeia de Markov	Usabilidade

Autores	Título	Local	Ano	Contribuição	Tipo Artigo	Local Testes	Modelos	Característica de Qualidade
Sprenkle *et al.*	Configuring effective navigation models and abstract test cases for web applications by analysing user behaviour	Software Testing Verification and Reliability	2013	Outros	Avaliação experimental	Servidor	Cadeia de Markov	Usabilidade
Tonella *et al.*	Finding the optimal balance between over and under approximation of models inferred from execution logs	International Conference on Software Testing, Verification and Validation	2012	Outros	Prova de conceito	Cliente/ Servidor	Outros	Não definido
Tonella *et al.*	Automated generation of state abstraction functions using data invariant inference	International Workshop on Automation of Software Test	2013	Outros	Prova de conceito	Cliente	Máquina de Estados	Não definido
Törsel	Automated test case generation for web applications from a domain specific model	International Computer Software and Applications Conference	2011	Técnica	Proposta de solução	Servidor	Workflow	Não definido
Törsel	A testing tool for web applications using a domain-specific modelling language and the NuSMV model checker	International Conference on Software Testing, Verification and Validation	2013	Técnica	Proposta de solução	Cliente/ Servidor	Máquina de Estados	Não definido
Xi *et al.*	An Approach to transforming UML model to FSM model for automatic testing	International Conference on Computer Science and Software Engineering	2008	Técnica	Proposta de solução	Não definido	Máquina de Estados	Não definido
Xiong *et al.*	Model-based penetration test framework for web applications using TTCN-3	Lecture Notes in Business Information Processing	2009	Técnica	Proposta de solução	Cliente/ Servidor	Outros	Segurança
Xu	A tool for automated test code generation from high-level Petri nets	Lecture Notes in Computer Science	2011	Ferramenta	Prova de conceito	Servidor	Redes de Petri	Segurança
Xu *et al.*	Automated security test generation with formal threat models	IEEE Transactions on Dependable and Secure Computing	2012	Técnica	Avaliação experimental	Cliente/ Servidor	Redes de Petri	Segurança
Xu *et al.*	Mining executable specifications of web applications from Selenium IDE tests	International Conference on Software Security and Reliability	2012	Técnica	Avaliação experimental	Cliente/ Servidor	Redes de Petri	Manutenibilidade
Zakonov e Shalyto	Extracting EFSMs of web applications for formal requirements specification	Lecture Notes in Computer Science	2012	Técnica e Ferramenta	Prova de conceito	Cliente/ Servidor	Máquina de Estados	Não definido
Zhang *et al.*	Introducing Test Case Derivation Techniques into Traditional Software Development: Obstacles and Potentialities	International Conference on Software Testing, Verification, and Validation	2011	Ferramenta	Proposta de solução	Servidor	Baseado em UML	Manutenibilidade
Zheng *et al.*	An automatic test case generation framework for web services	Journal of Software	2007	Técnica	Prova de conceito	Servidor	Outros	Funcionalidade

2.7.5. Análise dos Resultados

Nesta seção serão mostradas as análises dos resultados.

2.7.5.1. Ano de publicação

Os 57 artigos foram publicados entre 2005 e 2013. Na Figura 20, pode-se observar que no período de 2011 a 2013 ocorre a maior concentração de publicações relacionadas ao TBM em aplicações web, cujos valores são (13: 22,8%), (11: 19,3%) e (16: 28,1%), respectivamente.

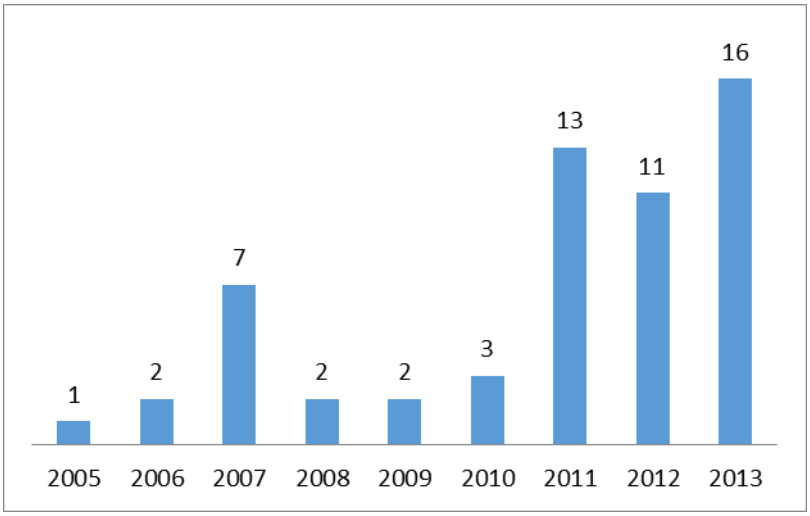

Figura 20: Ano de publicação

2.7.5.2. Local de publicação

Na Tabela 4, é mostrada a lista com os locais que obtiveram pelo menos duas publicações.

Tabela 4: Principais locais de publicação

Local de publicação	Acrônimo	Publicações
International Conference on Software Testing, Verification and Validation	ICST	8
Brazilian Workshop on Systematic and Automated Software Testing	SAST	4
International Workshop on Automation of Software Test	AST	2
International Computer Software and Applications Conference	COMPSAC	2
Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering	ESEC /FSE	2
Brazilian Symposium on Software Engineering	SBES	2
Brazilian Symposium on Software Quality	SBQS	2

Os locais com maior número de publicação foram: SAST (4: 7,0%) e ICST (8: 14,0%). Na categoria Outros estão 35 trabalhos (61,4%) que foram publicados em diferentes locais.

2.7.5.3. Tipo de contribuição

Na Figura 21 são mostrados os resultados por Tipo de contribuição, no qual pode-se verificar que a maior parte da contribuição de TBM em aplicações web destina-se a descrever técnicas (43: 75,4%), seguida por trabalhos que propõem uma ferramenta (14: 24,6%) para suporte parcial ou total a uma técnica[2]. Também são encontrados trabalhos que propõem modelos (2: 3,5%) ou processos (2: 3,5%).

Na categoria *Outros*, estão 7 trabalhos (12,3%) que tratam de comparação e avaliação de modelos.

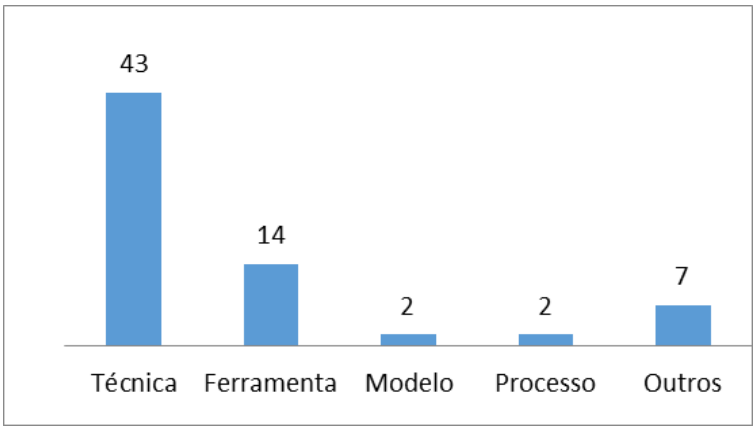

Figura 21: Tipo de contribuição

2.7.5.4. Tipo de estudo

Na Figura 22 são mostrados os resultados por Tipo de estudo, no qual pode-se observar que pesquisa em TBM em aplicações Web é dominada por propostas de solução (27: 49,1%) e prova de conceito (18: 31,6%). Já as avaliações experimentais apresentam 10 trabalhos (17,5%) e apenas dois trabalhos (3,5%) apresentam resultados na indústria.

[2] Descrita em um trabalho anterior ou no mesmo trabalho.

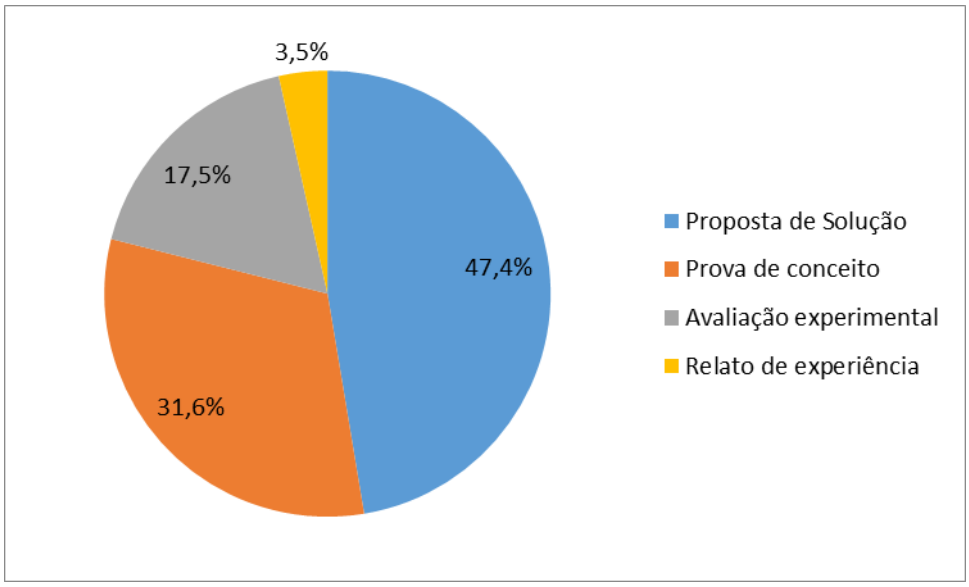

Figura 22: Tipo de Estudo

2.7.5.5. Localização do teste

Na Figura 23 são mostrados os resultados por Localização dos testes, no qual pode-se observar que a maior parte dos testes ocorrem no lado do servidor (34: 59,6%), seguidos por testes que abordam tanto o lado cliente quanto o lado servidor (17: 29,8%) e em terceiro lugar são mostrados os testes que ocorrem apenas no lado cliente (5: 8,8%). Em um dos trabalhos não é possível identificar a localização dos testes (1: 1,8%), pois o foco desse trabalho é comparar diferentes modelos ou técnicas.

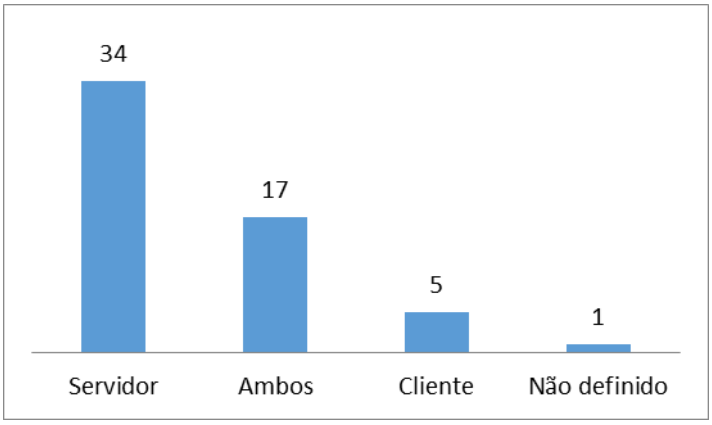

Figura 23: Localização dos testes

2.7.5.6. Modelos utilizados

Na Figura 24 são mostrados os resultados por Modelos utilizados, no qual pode-se observar que os principais modelos que representam o

comportamento/estrutura do sistema, são Máquinas de Estados[3] e Diagramas UML, com 14 trabalhos (24,6%) cada um.

Dentre os Diagramas UML utilizados, podem ser citados os seguintes diagramas: Classe, Caso de uso, Atividade, Máquina de estados, Perfil de teste.

Já os Modelos baseados em *Workflow* apresentam 8 trabalhos (14,0%), nos quais podem ser citados: Modelo de fluxo de evento, grafo de sequência de eventos, dentre outros.

Modelos que utilizam Redes de Petri são abordados em 4 trabalhos (7,0%), enquanto que 3 (5,3%) trabalhos utilizam Cadeias de Markov.

Na categoria *Outros*, estão 14 trabalhos (24,6%) que referem-se aos modelos representados em *Object Z* e, ainda, alguns trabalhos possuem um modelo representando informações provenientes dos requisitos do software, mas o modelo é descrito em uma notação diferente da UML.

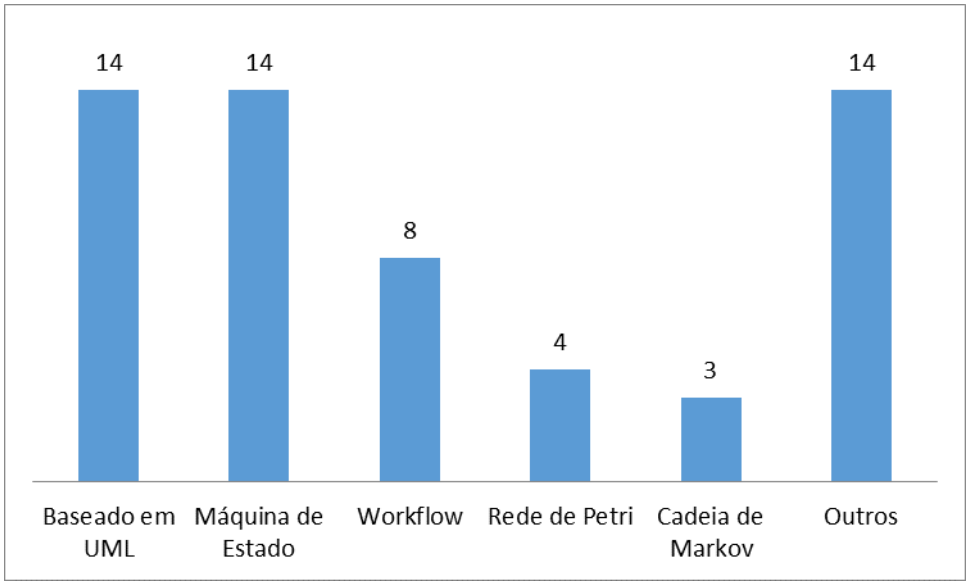

Figura 24: Modelos utilizados

2.7.5.7. Nível de teste

Na Figura 25 é possível observar que os testes de sistema têm chamado mais a atenção dos pesquisadores em TBM (43: 75,4%). Em seguida, estão os testes de Unidade (21: 36,8%), passando a Testes de Integração (8: 14,0%), Testes de

[3] Representadas por meio de máquinas de estados finitos (estendidas)

regressão[4] (6: 10,5%) e, por fim, Testes de Aceitação, com apenas 2 trabalhos (3,5%).

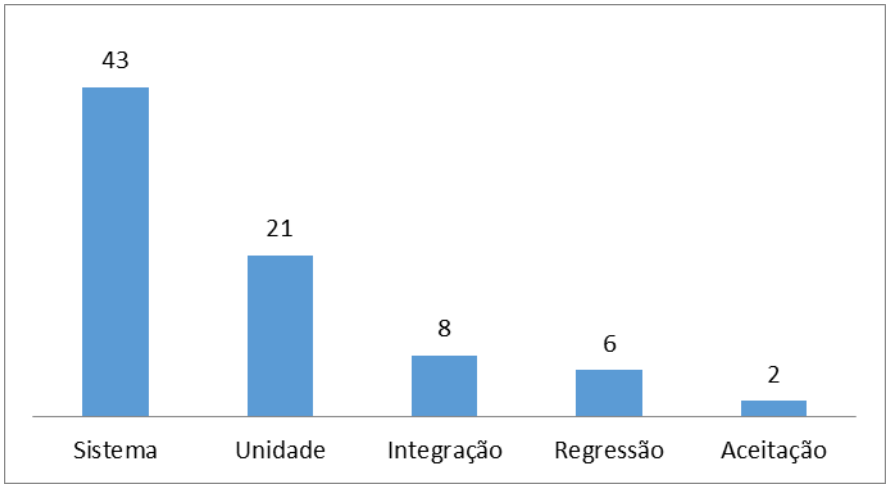

Figura 25: Níveis de teste

2.7.5.8. Características de qualidade

Na Figura 26 são mostrados os resultados por Características de qualidade voltadas para aplicações Web. As Funcionalidades da aplicação são verificadas em 33 trabalhos (57,9%), nos quais grande parte das abordagens verificam o fluxo de navegação das aplicações quanto ao processamento correto e a integridade dos links.

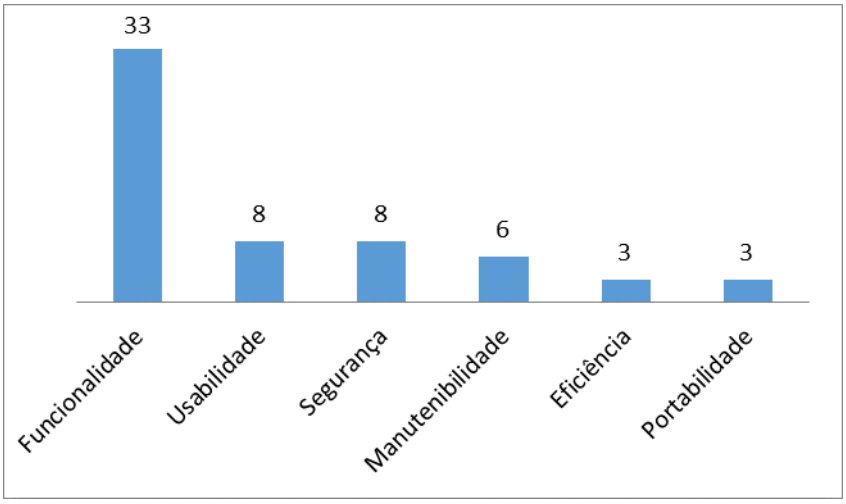

Figura 26: Características de qualidade

Em 8 trabalhos (14,0%) o foco é Usabilidade, verificando-se a facilidade no uso da aplicação e aprendizagem das terminologias, menus e navegação por meio

[4] Não se refere a um Nível de Teste exatamente, porém algumas técnicas foram desenvolvidas para apoiar esta estratégia de re-execução de teste.

de diferentes páginas da Web. Em outros 8 trabalhos (14,0%), a preocupação é a segurança das aplicações, nos quais são avaliados o controle de acesso, vulnerabilidades da aplicação e sua capacidade de evitar “ataques” por meio da injeção de códigos maliciosos.

Foram identificados 6 trabalhos (10,5%) que tratam da manutenibilidade das aplicações. A eficiência e a portabilidade da aplicação são tratadas em 3 trabalhos (5,3%), cada uma.

2.7.5.9. Existência de Apoio Ferramental

Nesta categoria foram verificados se os trabalhos produziam alguma ferramenta de apoio, cujo resultado é apresentado na Figura 27. Foram identificados 20 trabalhos (35,1%) que ofereciam suporte ferramental total ou parcial da metodologia empregada, enquanto que 37 trabalhos (67,9%) não descrevem apoio ferramental.

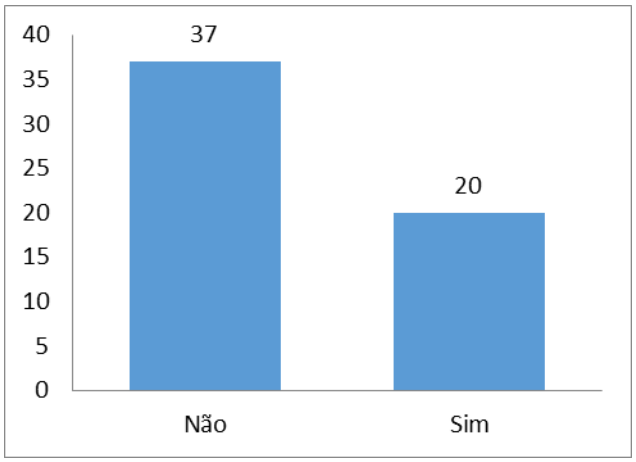

Figura 27: Gera ferramentas

Além da verificação dos trabalhos que possuíam apoio ferramental, também verificou-se que a ferramenta *Guitar* foi citada em 6 trabalhos (10,5%) como trabalho relacionado e reconhecida como o estado da arte em teste GUI, por isso a mesma foi escolhida para ser o objeto desta pesquisa.

2.7 Considerações Finais

Neste capítulo foram apresentados os principais conceitos necessários ao entendimento e desenvolvimento desta pesquisa, os trabalhos relacionados e os

resultados do Mapeamento Sistemático realizado. No próximo capítulo será apresentado processo de evolução da ferramenta *Web Guitar*, desde as limitações identificadas nesta ferramenta até as mudanças realizadas, produzindo uma nova versão que viabilize a geração de um conjunto mais amplo de casos de teste, objetivo principal deste trabalho.

CAPÍTULO 3 – EVOLUÇÃO DA FERRAMENTA WEB *GUITAR*

Neste capítulo é descrito o processo de adaptação da ferramenta Web Guitar para melhoria na qualidade do modelo GUI utilizado na geração de casos de teste. Também são detalhadas as dificuldades encontradas na utilização da ferramenta, bem como as limitações tratadas e não tratadas.

3.1. Introdução

O Mapeamento Sistemático descrito no Capítulo 2 sobre TBM aplicado a Aplicações Web revelou uma concentração de trabalhos com pouca evidência empírica e pouca aplicação na indústria. Esse fato contribuiu para o direcionamento desta pesquisa, pois ao invés de se desenvolver mais uma ferramenta ou *framework* para teste GUI em aplicações Web, optou-se em buscar uma ferramenta existente na academia e adaptá-la com o objetivo de apoiar sua inserção na indústria. Ao serem identificadas ferramentas para teste GUI em aplicações Web descritas na literatura, a ferramenta *Guitar* foi escolhida devido sua ampla utilização em ambiente acadêmico (AHO *et al.*, 2011; MARIANI *et al.*, 2012). Nesta pesquisa, trabalha-se com a extensão da ferramenta *Guitar* para aplicações Web, denominada *Web Guitar*.

Para adequar a ferramenta *Web Guitar* foi necessário estudar e avaliar quais modificações eram necessárias, e para isso foram realizadas as seguintes atividades:

- Identificar limitações descritas na literatura técnica;
- Utilizar a ferramenta para identificar outras limitações;
- Priorizar as modificações na ferramenta*;*
- Implementar as modificações escolhidas.

3.2. Atividades realizadas

As atividades realizadas no desenvolvimento desta pesquisa serão detalhadas a seguir.

3.2.1. Identificar limitações descritas na literatura

Foram identificadas as seguintes limitações descritas na literatura:

- AHO *et al.* (2011) relatam dificuldades em usar a ferramenta *Guitar* para gerar modelos de aplicações Java GUI complexas.
- MARIANI *et al.* (2012) ressaltam problemas da ferramenta *Guitar* ao interagir com algumas janelas frequentemente utilizadas, como janela de login;
- NGUYEN *et al.* (2013) descrevem várias restrições da ferramenta *Guitar*, todas relacionadas ao módulo *Ripper,* as quais podemos citar:
    1. A execução do algoritmo do *Web Ripper* dentro de uma única instância da aplicação leva a geração de árvores GUI imprecisas;
    2. A árvore GUI gerada pelo *Web Ripper* requer validação manual;
    3. Problemas na identificação de componentes GUI durante a exploração da aplicação também pode levar a geração de árvores GUI imprecisas.

3.2.2. Utilização da ferramenta

Ao configurar e instalar a ferramenta *Web Guitar*, foram encontradas as seguintes restrições relacionadas ao seu ambiente de execução:

- Execução apenas no navegador Firefox versão 6;
- Execução apenas em máquina Linux 32 bits.

Para entender as limitações relatadas na literatura, foram realizados testes em pequenas aplicações Web interativas e transacionais e observou-se que:

O *Web Ripper* guarda informações da URL da página para saber se uma determinada página já foi visitada. Quando ele detecta uma página pela primeira vez, ele percorre a mesma, porém nas próximas vezes que ele a encontra, essa página é ignorada. Esse problema se repetiu várias vezes, tornando-se evidente em aplicações, cujas páginas possuíam mais de um caminho de execução ou que eram utilizadas em mais de uma funcionalidade, como no exemplo que será apresentado na Figura 28 e Figura 29. Nessas figuras são mostradas duas variações da página *index.php* da aplicação *Phone Book*, na Figura 28, a página *index.php* é usada para mostrar a lista de todos os contatos, já na Figura 29 essa página é usada para cadastrar um novo contato. Nesse exemplo, somente da página da Figura 28 era visitada, enquanto que a segunda variação desta página era ignorada.

Figura 28: Página index.php (1) do *Phone Book*

Figura 29: Página index.php (2) do *Phone Book*

Foram observados problemas na identificação do componente *<select>,* o qual é comum em aplicações web. O *Web Ripper* percorria o componente, porém não conseguia gerar um caso de teste que contemplasse o mesmo. Logo, observou-se a necessidade de estender o suporte a esse componente.

Em relação a árvore GUI, constatou-se que o seu tamanho tornava trabalhosa sua manipulação e busca por informações.

Utilizando-se a ferramenta *Web Guitar* para inserir valores em campos de entrada, foram observadas as seguintes dificuldades:

- O código fonte disponibilizado não fornecia suporte as entradas manuais;
- Criação do arquivo de configuração exigia grande esforço.

Observou-se diferenças entre a descrição da ferramenta *Guitar* em NGUYEN *et al.* (2013) e a implementação da *Web Guitar*. Foi averiguado que o código fonte disponibilizado necessitava de ajustes para permitir a inserção de entradas manuais. Em consequência disto, houve dificuldade em se criar o arquivo de configuração para inserir informações das entradas manuais, visto que era necessário conhecer previamente tanto a estrutura desse arquivo *XML* quanto a estrutura da página.

3.2.3. Priorizar modificações

A ideia de melhorar a ferramenta *Web Guitar* para facilitar sua adoção na indústria, conduziu a priorização dos seguintes pontos:

- Dentre as limitações da *Web Guitar* descritas na literatura, o uso de apenas uma instância da página foi escolhida como sua principal limitação, visto que seriam excluídas as aplicações Web que possuíssem mais de um fluxo de execução por páginas e/ou aplicações que reutilizassem páginas em várias funcionalidades.
- Melhoria do suporte as entradas manuais para reduzir as seguintes dificuldades constatadas:
  a) diversos componentes eram gerados em tempo de execução e muitas vezes esses componentes dependiam de informações fornecidas pelos usuários;
  b) determinadas entradas eram fundamentais para acessar funcionalidades das aplicações, principalmente aquelas que possuíam *login*;

c) esforço e tempo extra gasto na elaboração do arquivo de configuração.

As limitações tratadas juntamente com os detalhes de suas implementações serão apresentadas na próxima seção.

3.3. Modificações realizadas

As alterações realizadas na ferramenta *Web Guitar* são apresentadas na Figura 30, as quais encontram-se destacadas. Como pode ser observado, foram realizadas modificações nos módulos *Web Ripper* e *Web Replayer.* No primeiro módulo, foi inserido um arquivo de configuração de saída a cada execução do *Web Ripper*. Na próxima seção serão detalhadas as modificações inseridas.

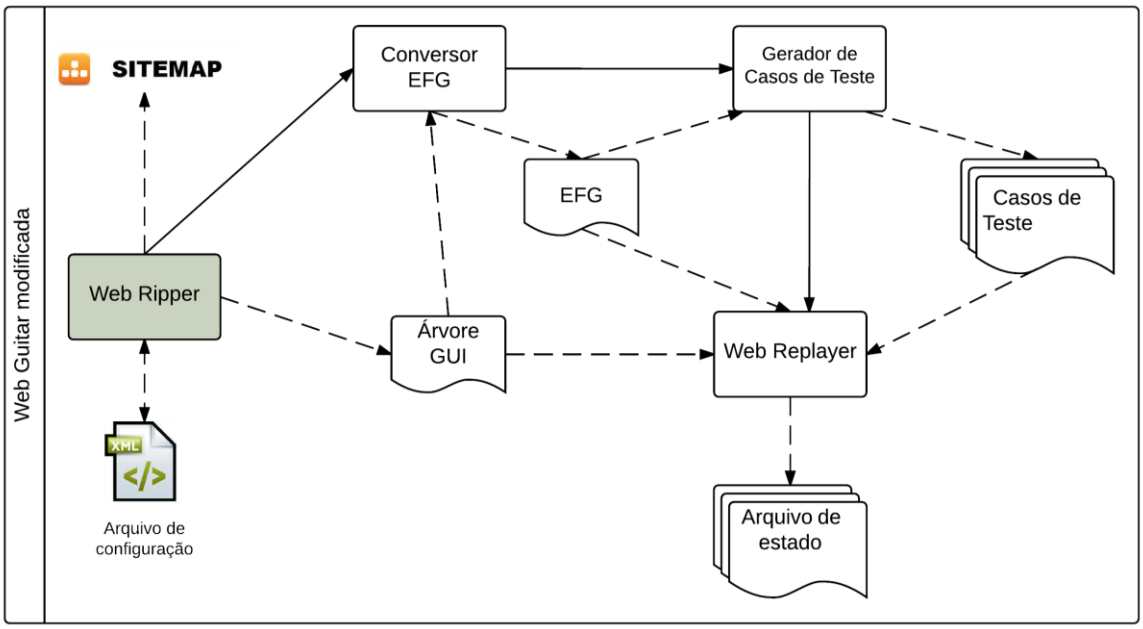

Figura 30: Web Guitar modificada

3.3.1. Valores de entrada

O arquivo de configuração (*configuration.xml*) na versão original do *Web Ripper* deveria receber os componentes terminais e as páginas ignoradas. Ao verificar que esse arquivo também poderia receber valores de campos de entrada, então o *Web Ripper* foi modificado para que ele gerasse o arquivo de configuração com as páginas percorridas e os campos de entradas dessas páginas, que será mostrado utilizando a página *create.php* que permite cadastrar um novo estudante no SIS3.

Quando o *Web Ripper* modificado percorre a aplicação que contém a página *create.php* (Figura 31), ele identifica os componentes de entrada por meio de IDs ou *tagName* do componente. No final da sua execução, o *Web Ripper* gera um arquivo de configuração, denominado *configurationOutput.xml.* Nesse arquivo estão todos os componentes de entrada identificados, sem incluir seus valores, conforme a Figura 32, que apresenta parte do arquivo *configurationOutput.xml.* Componentes que não possuam ID ou *tagName* não podem ser identificados.

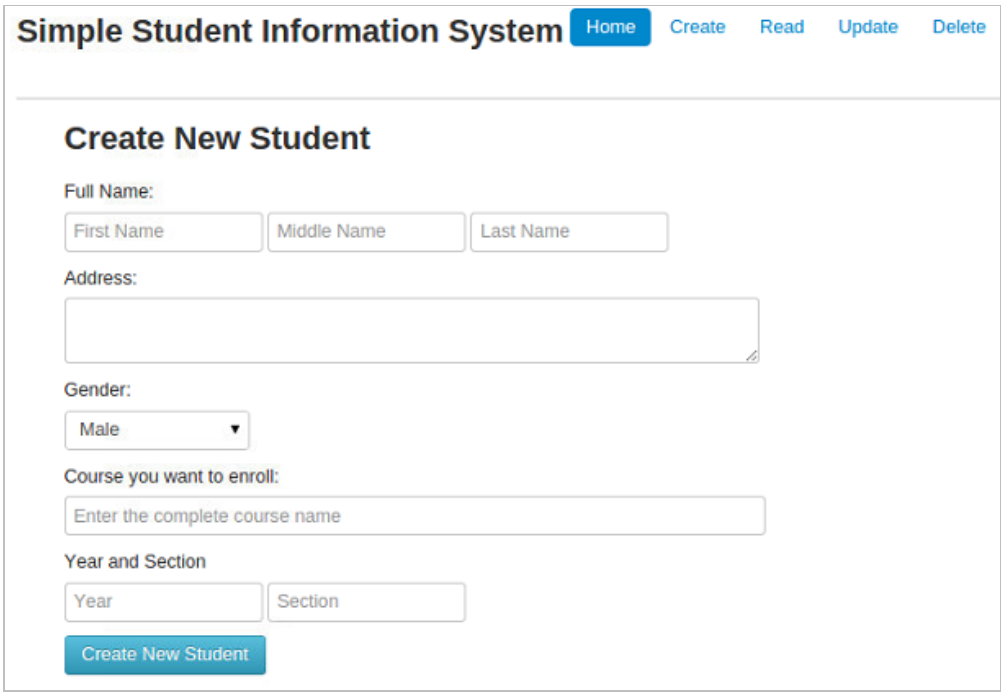

Figura 31: Página *create.php*

```
<FullComponent>
    <Window>
        <Attributes>
            <Property>
                <Name>Title</Name>
                <Value>http://localhost/SIS3/create.php</Value>
            </Property>
        </Attributes>
    </Window>
    <Component>
       <Attributes>
            <Property>
                <Name>fname</Name>
                <Value></Value>
            </Property>
            <Property>
                <Name>mname</Name>
                 <Value></Value>
            </Property>
            <Property>
                <Name>lname</Name>
                 <Value></Value>
            </Property>
            <Property>
                <Name>addr</Name>
                 <Value></Value>
            </Property>
            <Property>
                <Name>gender</Name>
                  <Value></Value>
            </Property>
            <Property>
                <Name>course</Name>
                <Value></Value>
            </Property>
```

Figura 32: Parte do arquivo *configurationOutput.xml* referente a página *create.php*

Para utilizar os dados de entrada, deve-se editar o arquivo de configuração e incluir os valores desejados, conforme Figura 33.

```
<FullComponent>
    <Window>
        <Attributes>
            <Property>
                <Name>Title</Name>
                <Value>http://localhost/SIS3/create.php</Value>
            </Property>
        </Attributes>
    </Window>
    <Component>
       <Attributes>
            <Property>
                <Name>fname</Name>
                <Value>Carlos</Value>
            </Property>
            <Property>
                <Name>mname</Name>
                 <Value>Domingues</Value>
            </Property>
            <Property>
                <Name>lname</Name>
                 <Value>Ascate</Value>
            </Property>
            <Property>
                <Name>addr</Name>
                 <Value>Rua 04, 48 - Bairro da Paz</Value>
            </Property>
            <Property>
                <Name>gender</Name>
                  <Value>Male</Value>
            </Property>
            <Property>
                <Name>course</Name>
                <Value>Engenharia de Produção</Value>
            </Property>
```

Figura 33: Arquivo *configurationOutput.xml* da página *create.php* editado

Assim, na segunda execução o *Web Ripper* preenche as páginas com os dados de entradas do arquivo de configuração editado (Figura 34). Desta maneira, ao se criar a estrutura do arquivo de configuração, o testador não precisaria criar toda a estrutura desse arquivo, ele apenas editaria manualmente os valores dos campos de entrada.

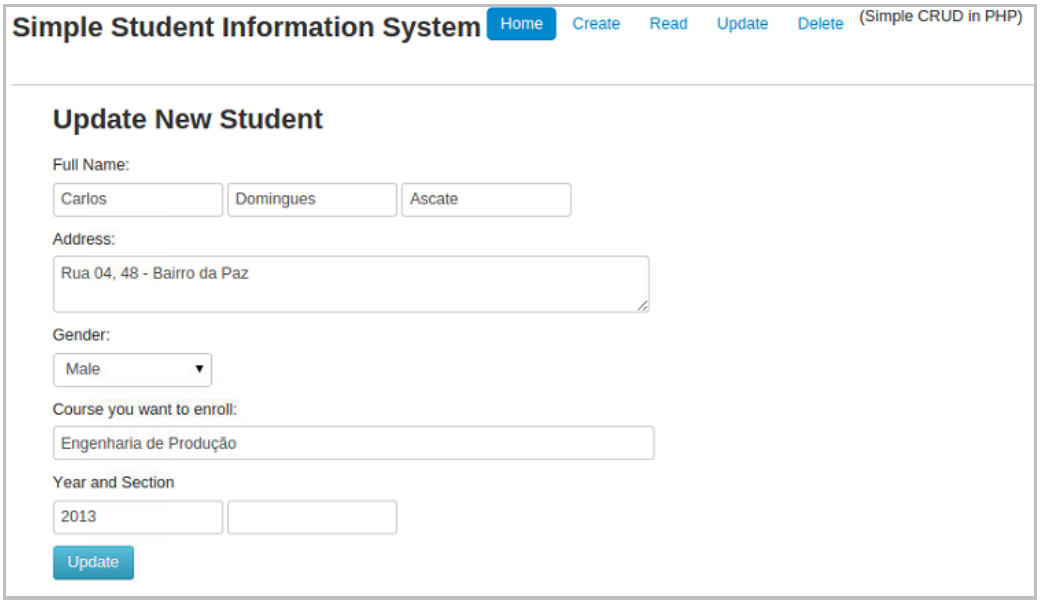

Figura 34: Página *create.php* preenchida

Os dados de entrada que anteriormente só eram destinados a campos de texto foram estendidos também a componentes *checkbox* e *radiobutton*, pois verificou-se que esses componentes são usados para renderizar outros elementos, como será mostrado no exemplo a seguir utilizando a página Nova Ocorrência que permite cadastrar uma ocorrência de trânsito que deverá ser tratada pelo órgão competente. Na Figura 35, essa página é mostrada conforme a mesma é carregada pelo navegador Web. Ao marcar-se o *checkbox* Novo Solicitante, ele oculta os componentes destacados na Figura 35 e exibe outros três componentes destacados na Figura 36. Nesse exemplo ao se alterar os valores do *checkbox* Novo Solicitante é possível simular esses dois comportamentos da página Nova Ocorrência.

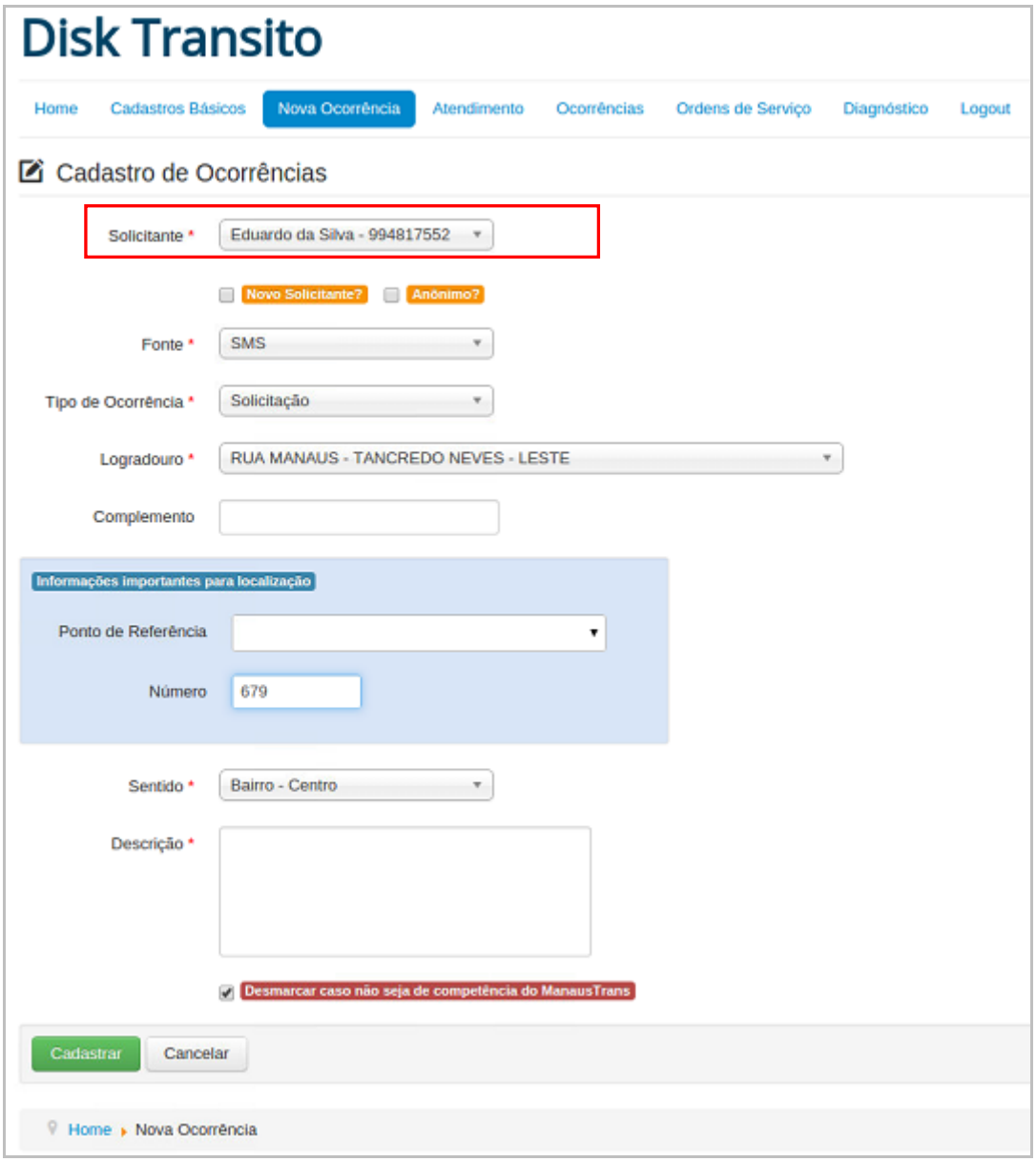

Figura 35: Página Nova Ocorrência – Instância 1

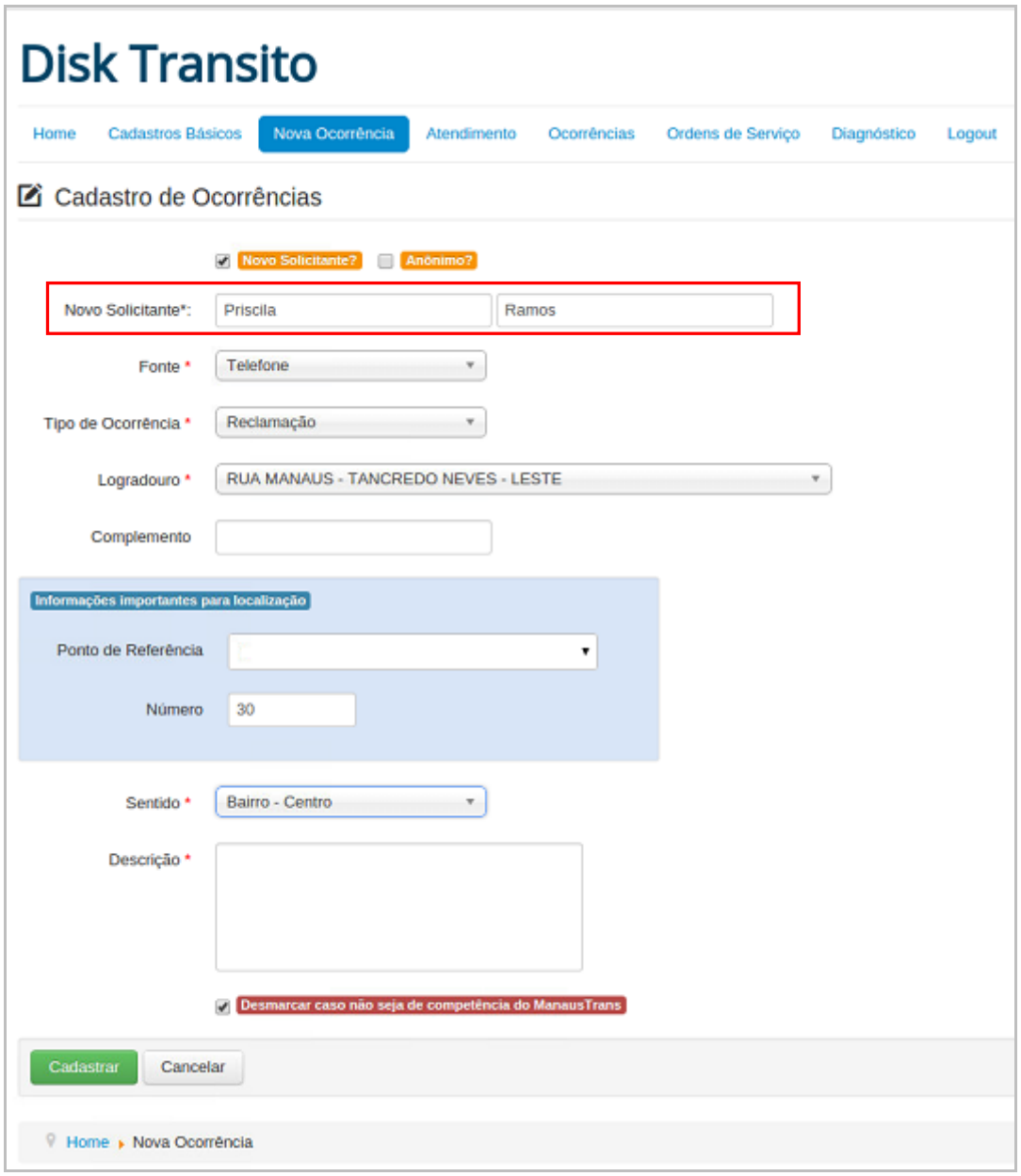

Figura 36: Página Nova Ocorrência – Instância 2

Para executar as modificações descritas nessa seção foram realizadas as modificações no código apresentadas na Figura 37.

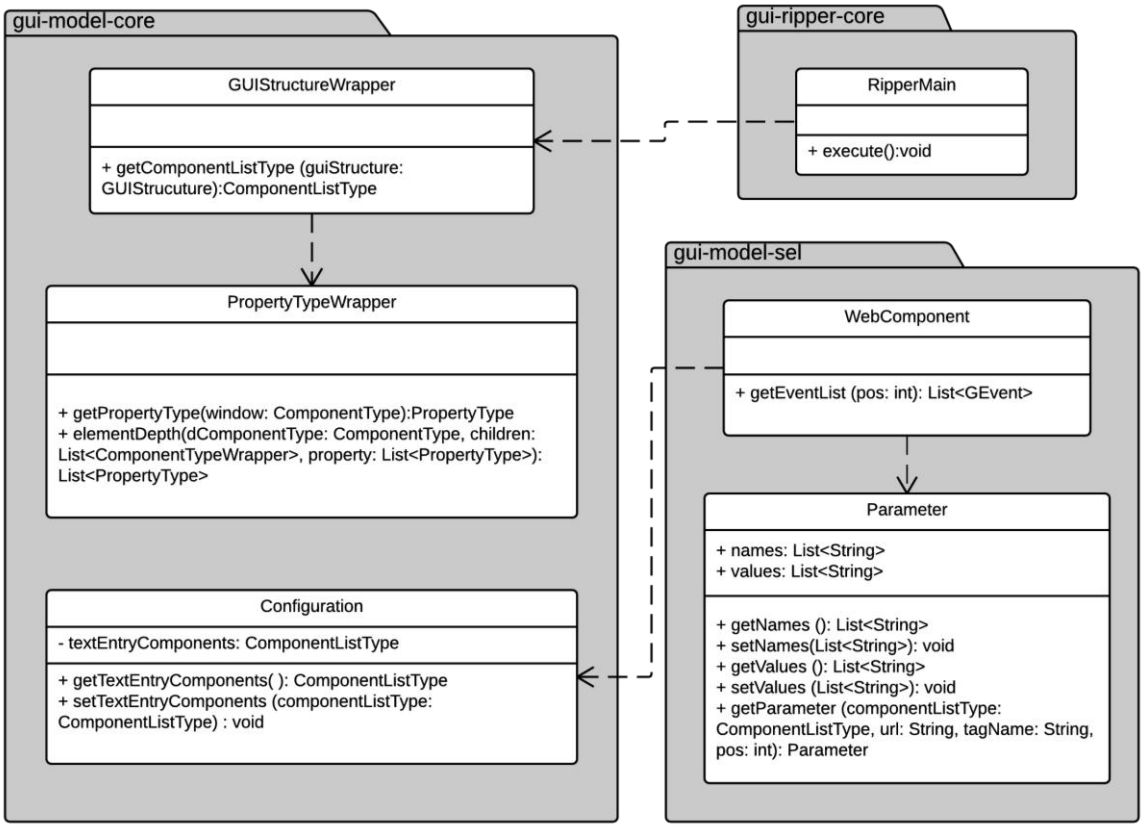

Figura 37: Modificações realizadas no código para inserir valores de entrada

O método *execute da* classe *RipperMain passou a gerar o arquivo configurationOutuput.*

A classe *Parameter* foi criada para manipular o conjunto de entradas, por isso contém como atributos um conjunto de nomes e valores dos dados de entrada.

Para permitir o recebimento dos dados de entrada, na classe *Configuration* foi incluído um atributo *textEntryComponents* que representa o conjunto dos campos de entradas contidos no arquivo de configuração.

Em seguida foram incluídos novos métodos as classes *GuiStructureWrapper* e PropertyTypeWrapper para converter um objeto *Configuration* em um objeto *Parameter.* Esse objeto é utilizado no método *getEventList* da classe *WebComponent* para preencher os dados de entrada no momento em que está sendo feita a exploração da aplicação.

3.3.2. Inclusão de várias instâncias

Como o *Web Ripper* original trabalha apenas com uma instância para cada página visitada, foi necessário incluir o conceito de várias instâncias. Essa mudança foi realizada tendo em vista que determinadas páginas possuem várias variações, como no exemplo mostrado acima e na Figura 38 e Figura 39, em que são apresentadas duas instâncias da página *Dados do Aluno* do Sistema de Apoio ao PPGI/UFAM, uma referente a um aluno de mestrado (que possui apenas uma qualificação e uma defesa final de dissertação) e outra referente a um aluno de doutorado (que possui duas qualificações, além da defesa final de tese).

Figura 38: Instância 1 da página Dados do Aluno

Figura 39: Instância 2 da página Dados do Aluno

Para que fosse possível utilizar o conceito de várias instâncias, foi necessário estudar uma forma de diferenciar uma instância das demais instâncias em uma página. Inicialmente, a forma mais simples para diferenciar instâncias de páginas foi por meio da quantidade de *tags* HTML. Para isso, bastava verificar a quantidades de componentes em cada página visitada. O problema com esse método era que aplicações dinâmicas renderizam seus campos continuamente e era necessário atualizar a quantidade de componentes a cada alteração da GUI.

Quando o *Web Ripper* modificado explorava uma aplicação, ele permitia explorar várias instâncias de uma página, desde que a quantidade de *tags* HTML fosse diferente. Essa restrição limitava variações de página que possuíam a mesma quantidade de *tags* HTML.

Nesse caso, era preciso buscar uma outra maneira de diferenciar instâncias de uma página. Uma outra forma de distinguir instâncias de uma mesma página foi utilizando o conceito de estado da página, que já existia na *Web Guitar* original, porém não era utilizado para este fim. Este método foi escolhido pois não apresentava a restrição do método anterior.

Com as alterações no código fonte, o conceito de várias instâncias foi incorporado à ferramenta *Web Guitar*. Desta forma, quando o *Web Ripper* explora uma página, ele guarda informações a respeito do número da instância e do estado da página explorada. Ao encontrá-la novamente, ele verifica seu estado. Caso seja diferente do estado da página anterior, ele inclui essa nova instância na árvore GUI que armazena o estado da aplicação.

No exemplo a seguir, o *Web Ripper* original só passaria pela primeira instância da página Dados do Aluno (Figura 38) e ignoraria a segunda instância dessa página. Já o *Web Ripper* modificado, uma vez configurado com valor da instância maior ou igual a 2 consegue explorar e distinguir a segunda instância da página inicial (mostrada na Figura 39).

Para que seja possível explorar mais de uma instância de página, foi incluído o parâmetro *–i*, seguido pela quantidade de instâncias desejadas na linha de comandos do Linux (Figura 40). Quando o parâmetro *-i* não é informado, o valor atribuído será 2.

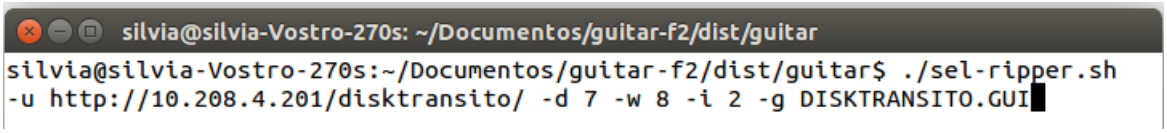

Figura 40: Chamada ao *Web Ripper* com inserção do parâmetro de número de instâncias (*-i*) e número de visitas (*-v*).

É importante ressaltar que cada instância encontrada será um novo componente <GUI> dentro da árvore GUI gerada.

Para executar as modificações descritas nessa seção foram realizadas as modificações no código apresentadas na Figura 41.

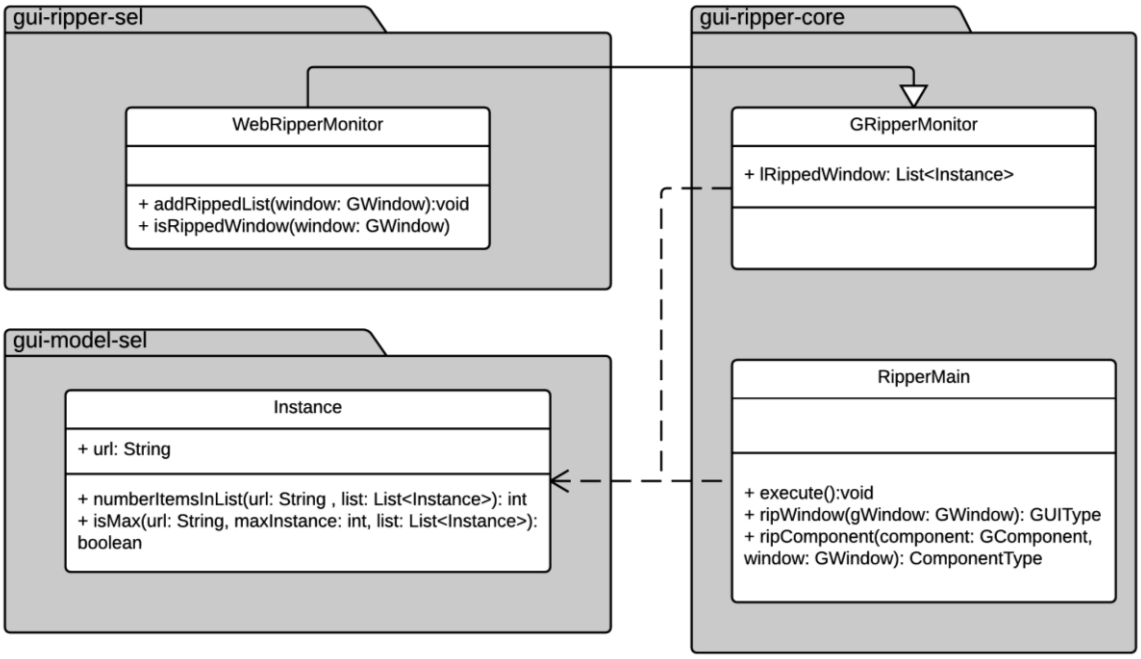

Figura 41: Modificações realizadas no código para trabalhar com várias instâncias

Primeiramente foi alterado o tipo do atributo *lRippedWindow* na classe GRipperWindow. Esse atributo contém a lista das páginas exploradas e anteriormente não permitia que páginas com a mesma URL fossem incluídas várias vezes.

Na classe *WebRipperMonitor* foram alterados o método *addRippedList* que adiciona páginas na lista de páginas exploradas e o método *isRippedWindow* que valida se uma página deve ser explorada ou não.

Para trabalhar com várias instâncias foi criada a classe *Instance* que representa cada instância de uma página. Nessa classe foram incluído o método *numberItemsInList* que verificar a quantidade de instâncias de uma página passada por parâmetro e o método *isMax* que valida se o número máximo de instâncias foi atingido.

3.3.3. Inclusão de número máximo de visitas

Outro conceito introduzido por meio da alteração na ferramenta *Web Guitar* foi o conceito de número máximo de visitas. Essa informação se refere a quantidade de vezes que uma página poderá ser explorada e é necessária para se estabelecer uma condição de parada, caso o *Web Ripper* não encontre o número de instâncias desejado. Essa informação será utilizada caso não se tenha encontrado o número máximo de instâncias. Por exemplo, se é informado o número de instâncias 3, porém

só existem duas instâncias da mesma página, ao se informar o número de visitas 5, o *Web Ripper* só irá visitar a mesma página cinco vezes, caso ele não tenha encontrado as três instâncias que buscava.

Para que seja possível limitar o número de visitas a uma página, foi incluído o parâmetro *-v* seguido pelo número máximo de visitas permitidas na linha de comandos do Linux (Figura 40). Quando o parâmetro *-v* não é informado, o valor atribuído será duas vezes o número de instâncias (parâmetro *–i*, informado anteriormente). É importante ressaltar que esses valores devem ser ajustados para garantir melhor eficiência do Web Ripper dentro da Web *Guitar Modificada*.

Para incluir o conceito de número máximo de visitas foi realizada a modificação no código apresentada na Figura 42. O método *isRippedWindow* da classe *WebRipperMonitor* foi alterado para verificar se o número máximo de visitas foi atingido, anteriormente ele só verificava se o número de instancias havia sido atingido.

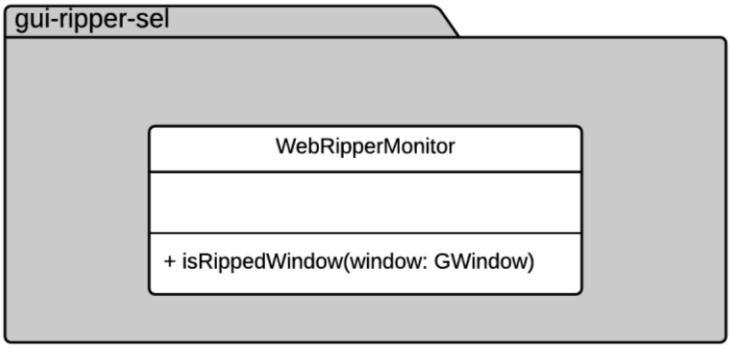

Figura 42: Modificação realizada no código para se incluir o número máximo de visitas

3.3.4. Outras modificações

Foram realizadas ainda as seguintes alterações:

- Inclusão de suporte ao componente <*select*>, que não existia anteriormente.
- A página inicial, a partir da qual o *Web Ripper* é executado, foi substituída a tela exibida na Figura 11 pela página apresentada na Figura 43. Dessa forma, o *Web Ripper* fará a chamada a página local ao invés de chamar a página definida pelos autores da *Guitar*.

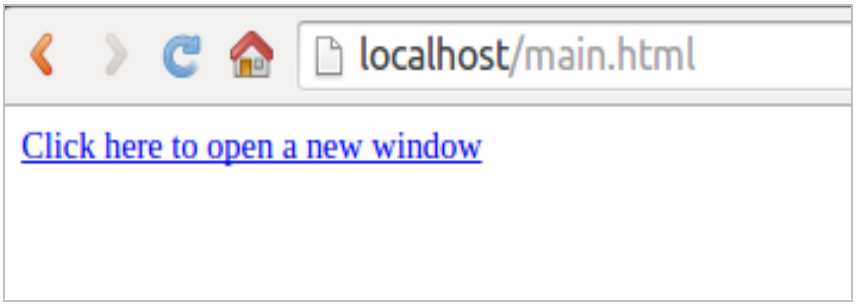

Figura 43: Página main.html

3.4. Limitações identificadas e não tratadas

Os problemas a seguir foram identificados, porém não foram tratados nesta pesquisa:

- Estender suporte a outros componentes Web mais complexos. Seria necessário testar aplicações que utilizassem outros componentes. Porém, devido sua diversidade isso se tornaria inviável. Exemplos desses componentes são mostrados na Figura 44: componente *AutoComplete*, que sugere valores a partir de dados digitados pelo usuário, e componente *PickList*, que permite mover dados de uma lista para outra.

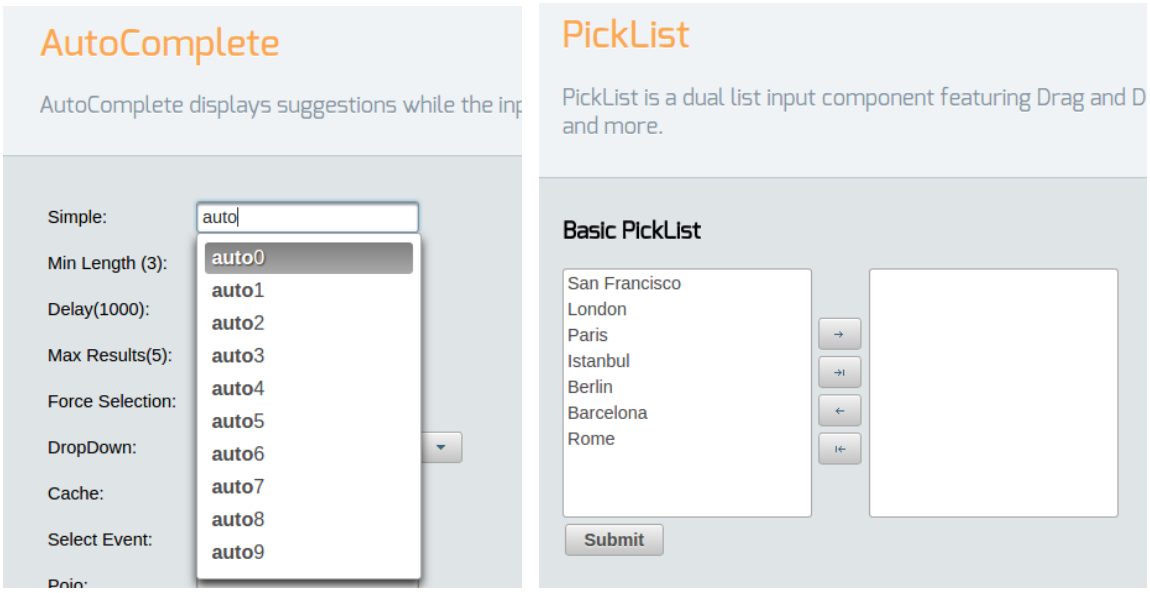

Figura 44: Componentes *Autocomplete* (esquerda) e *PickList* (direita).

- Como a árvore GUI está diretamente relacionada à quantidade de *tags* HTML, e as páginas Web possuem uma grande quantidade de *tags*, é difícil reduzir a quantidade de elementos da árvore *GUI.*

Existem outras limitações que podem dificultar a adoção do *Web Guitar pela* indústria e que não foram tratadas nesse momento, visto que o estudo de viabilidade foi executado em ambiente acadêmico. Essas restrições são relacionadas às características do sistema operacional (somente Linux 32 bits) e do navegador (Firefox 6) utilizados pela ferramenta. Acredita-se que as restrições citadas precisam ser resolvidas para que esta ferramenta seja aplicada na indústria, visto que essas restrições estão relacionadas à infraestrutura necessária para sua execução.

No próximo capítulo, será apresentado um estudo que teve como objetivo avaliar a viabilidade da versão evoluída da ferramenta *Web Guitar* em relação à geração de novos casos de testes não tratados pela sua versão original.

CAPÍTULO 4 - ESTUDO DE VIABILIDADE PARA AVALIAÇÃO DA CUSTOMIZAÇÃO DA *WEB GUITAR*

Este capítulo descreve o estudo de viabilidade realizado seguindo uma metodologia científica para avaliação de tecnologias de software, que consiste em uma seção de teste GUI utilizando as duas versões da ferramenta Web Guitar (original e modificada) que tem como objetivo gerar casos de teste, comparar os resultados de ambas as versões e avaliar sua eficiência e eficácia.

4.1. Introdução

Após implementação das modificações na ferramenta *Web Guitar*, o próximo passo foi realizar um estudo para avaliar a utilização da ferramenta proposta em relação à sua capacidade de gerar casos de teste com o propósito de amadurecê-la até que seja possível transferi-la ao contexto industrial. O objetivo deste estudo de viabilidade foi caracterizar a utilização da ferramenta *Web Guitar* modificada em relação à sua eficiência e eficácia para gerar e executar casos de teste em aplicações Web. Neste capítulo, a ferramenta *Web Guitar* original será denominada (*WG-Original)* enquanto a versão proposta neste trabalho será denominada (*WG-Modificada)*.

4.2. Definição do Estudo de Viabilidade

O objetivo do estudo de viabilidade é fornecer ao pesquisador informações suficientes para justificar (ou não) a continuação do seu trabalho (SHULL, 2001).

4.2.1. Propósito

O propósito do estudo foi responder a questão:

"A aplicação da evolução da ferramenta *Web Guitar* para geração e execução de casos de teste é viável, analisando sua eficiência e eficácia?"

Como detalhado no Capítulo 3, a ferramenta *WG-Original* necessitava de ajustes para receber os valores de entrada. Assim, para que houvesse uma comparação justa entre as duas versões da ferramenta, foi necessário incluir este ajuste na ferramenta *WG-Original*.

4.2.2. Perspectiva

A perspectiva é do ponto de vista dos pesquisadores, que desejam analisar a viabilidade da aplicação da ferramenta proposta.

- **Analisar** a utilização das ferramentas *WG-Original* e *WG-Modificada* na geração automática de casos de teste;
- **Com o propósito de** caracterizar;
- **Com respeito a** eficácia e eficiência das ferramentas em gerar e executar casos de teste;
- **Do ponto de vista do** pesquisador;
- **No contexto de** aplicações Web desenvolvidas por estudantes de graduação da UFAM.

4.2.3. Questões e Métricas

As questões abaixo se referem à aplicação da ferramenta *WG-Original e* ferramenta *WG-Modificada* para geração e execução de casos de teste em aplicações Web. A saber:

- Q1: Qual é a eficácia da utilização das versões *WG-Original* e *WG-Modificada* no que diz respeito à geração de casos de teste?

  Métricas:

  - $CT_{WG\text{-}Original}$ = Quantidade de casos de teste gerados pela ferramenta *WG-Original.*
  - $CT_{WG\text{-}Modificado}$ = Quantidade de casos de teste gerados pela ferramenta *WG-Modificada.*

- Q2: Qual é a eficiência da aplicação das versões *WG-Original* e *WG-Modificada* no que diz respeito à geração de casos de teste?

  Métricas:

  - $T_{WG\text{-}Original}$ = Tempo de execução do *Web Ripper* da WG-*Original.*

- Eficiência$_{(WG\text{-}Original)}$ = $CT_{WG\text{-}Original}$ / $T_{WG\text{-}Original}$.
- $T_{WG\text{-}Modificada}$ = Tempo de execução do *Web Ripper* da WG-Modificada.
- Eficiência$_{(WG\text{-}Modificada)}$ = $CT_{WG\text{-}Modificada}$ / $T_{WG\text{-}Modificada}$.

4.3. Planejamento do Estudo

Esse estudo teve o objetivo de geração e execução de casos de teste de aplicações Web reais por meio da utilização de duas versões da ferramenta *Web Guitar* (*WG-Original* e *WG-Modificada*). Com os resultados obtidos a partir da realização desse estudo é possível comparar a versão evoluída da ferramenta (*WG-Modificada*) em relação aos resultados obtidos com a sua versão original (*WG-Original*). Com isso, neste estudo de viabilidade buscou-se adquirir conhecimento que permita avaliar a viabilidade de aplicação da ferramenta proposta.

Com os resultados obtidos neste estudo, espera-se fornecer subsídios que permitam evoluir a ferramenta proposta de forma a reduzir o esforço necessário para sua utilização quanto aumentar sua capacidade de encontrar defeitos.

4.3.1. Formulação de Hipóteses

Para avaliar a evolução da ferramenta *Web Guitar* foi realizada uma comparação entre a quantidade de casos de teste gerados pela *WG-Original* e a quantidade de casos de teste gerados pela *WG-Modificada*.

Deseja-se comparar a eficácia e a eficiência das ferramentas *WG-Original* e *WG-Modificada* na geração de casos de teste. A eficácia é medida pelo número de casos de teste gerados, enquanto a eficiência também inclui o tempo.

Hipótese nula (H0): Não existe diferença entre os resultados encontrados pela ferramenta *WG-Original* e pela ferramenta *WG-Modificada* em relação a geração de casos de teste em sistemas Web.

Para se obter resultados com o objetivo de refutar ou confirmar a hipótese H0, a eficácia e a eficiência das abordagens foram as características avaliadas. Portanto, para cada uma dessas duas características, as seguintes hipóteses foram definidas:

Eficácia:

- $CT_{WG\text{-}Original}$ = Quantidade de casos de teste gerados por *WG-Original*.
- $CT_{WG\text{-}Modificada}$ = Quantidade de defeitos encontrados por *WG-Modificada*.

- **Hipótese nula A** (H0 A): A ferramenta *WG-Modificada* permite gerar a mesma quantidade de casos de teste gerados pela ferramenta *WG-Original*.
  H0 A: $CT_{WG\text{-}Modificada} = CT_{WG\text{-}Original}$

- **Hipótese Alternativa 1 A (H1 A)**: A ferramenta *WG-Modificada* permite gerar mais casos de teste que aqueles encontrados pela ferramenta *WG-Original*.
  H1 A: $CT_{WG\text{-}Modificada} > CT_{WG\text{-}Original}$

- **Hipótese Alternativa 2 A (H2 A):** A ferramenta *WG-Modificada* permite gerar menos casos de teste que aqueles encontrados pela ferramenta *WG-Original*.
  H2 A: $CT_{WG\text{-}Modificada} < CT_{WG\text{-}Original}$

Eficiência

- $T_{WG\text{-}Original}$ = Tempo de execução do *Web Ripper* da *WG-Original*.
- $CT_{WG\text{-}Original}$ = Quantidade de casos de teste gerados por *WG-Original.*
- $T_{WG\text{-}Modificada}$ = Tempo de execução do *Web Ripper* da *WG-Modificada*.
- $CT_{WG\text{-}Modificada}$ = Quantidade de casos de teste gerados por *WG-Modificada*.

- **Hipótese nula B (H0 B):** A relação entre a quantidade de casos de teste gerados e o tempo de execução é a mesma para a *WG-Modificada* e a *WG-Original*.
  H0 B: $CT_{WG\text{-}Modificada} \div T_{WG\text{-}Modificada} \approx CT_{WG\text{-}Original} \div T_{WG\text{-}Original}$

- **Hipótese Alternativa 1 B (H1 B):** A relação entre a quantidade de casos de teste gerados e o tempo de execução utilizando a ferramenta *WG-Modificada* é maior que o da *WG-Original*.
  H1 B: $CT_{WG\text{-}Modificada} \div T_{WG\text{-}Modificada} > CT_{WG\text{-}Original} \div T_{WG\text{-}Original}$

- **Hipótese Alternativa 2 B (H2 B):** A relação entre a a quantidade de casos de teste gerados e o tempo de execução utilizando a ferramenta *WG-Modificada* é menor que o da *WG-Original*.
  H2 B: $CT_{WG\text{-}Modificada} \div T_{WG\text{-}Modificada} < CT_{WG\text{-}Original} \div T_{WG\text{-}Original}$

4.3.2. Projetos a serem testados

Para a execução dos experimentos, foi disponibilizado o código fonte de duas aplicações Web, denominadas Sistema de Apoio ao PPGI/UFAM e Disk Transito. Essas aplicações foram desenvolvidas em linguagem PHP utilizando o *framework* de desenvolvimento *Joomla*.

Na Figura 45 e Figura 46 podem-se observar métricas como, linhas de código, número de classes, comentários e duplicações extraídas da ferramenta de gerenciamento de qualidade de software, *SonarQube*[5], para ambos os projetos.

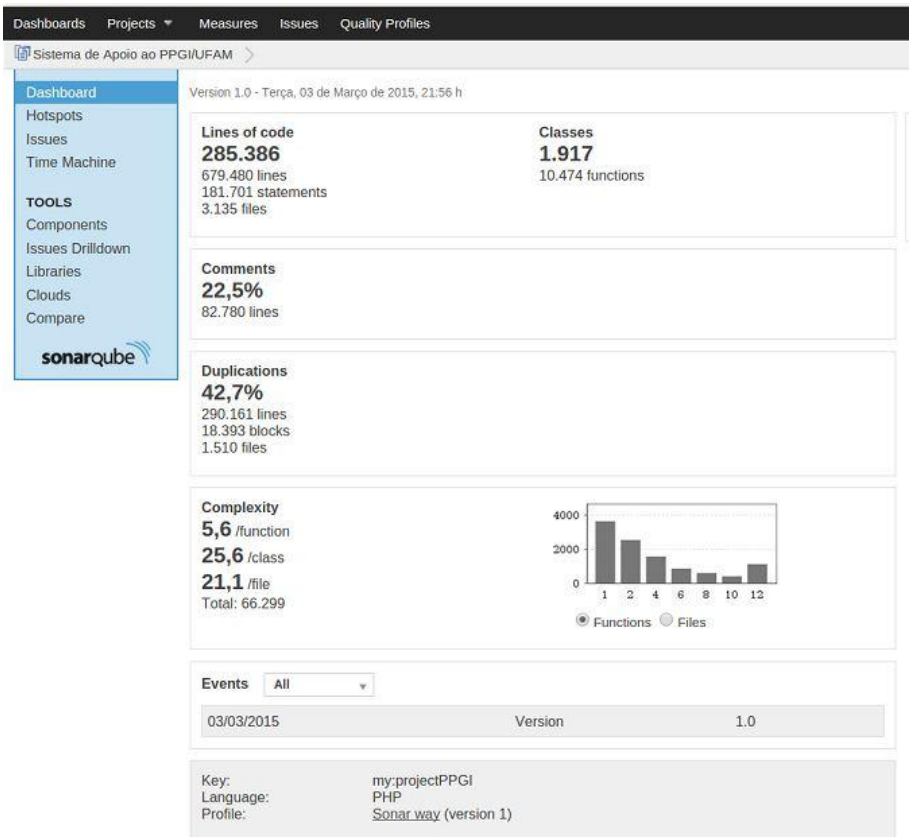

Figura 45: Métricas do Sistema de Apoio ao PPGI/UFAM

[5] http://www.sonarqube.org/

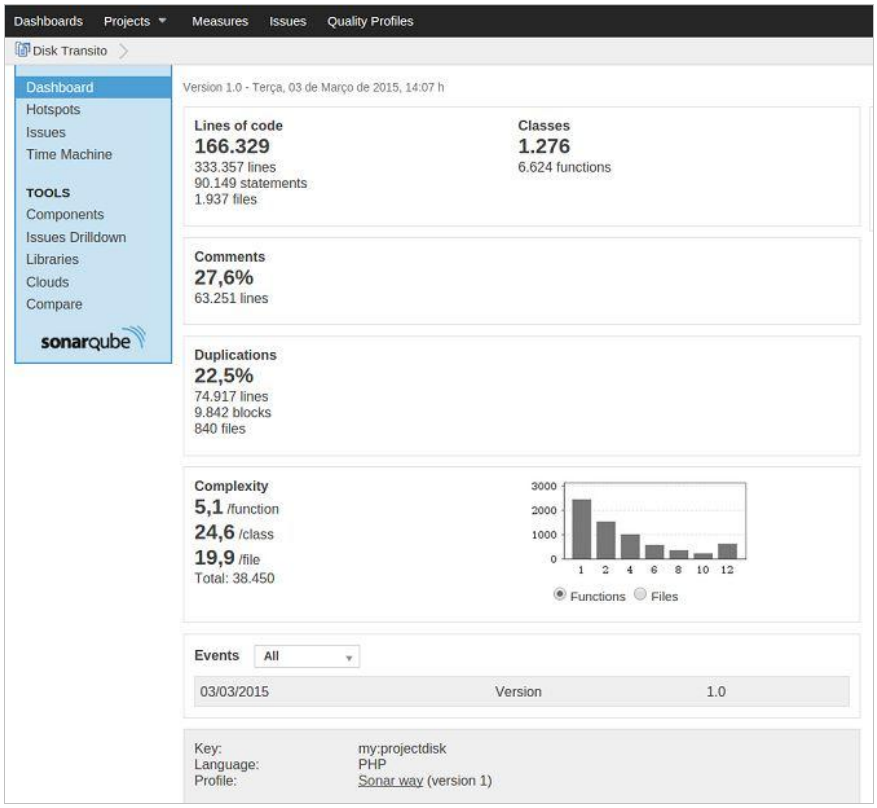

Figura 46: Métricas do Disk Transito

As métricas contidas nas Figura 45 e Figura 46 também estão resumidas na Tabela 5 em que pode-se verificar que o sistema de Apoio ao PPGI possui mais linhas de código e classes e também apresenta maior complexidade que o Disk Transito.

Tabela 5: Resumo de métricas do Sonarqube para os sistemas a serem testados

	Linhas de Código	Nº de classes	Duplicações	Complexidade por função	Complexidade por classe
Sistema de Apoio ao PPGI/AM	285.386	1.917	22,5%	5,6%	25,4%
Disk Transito	166.329	1.276	22,5%	5,1%	24,6%

Ambos os projetos são reais, originais e não foi inserido qualquer tipo de defeito pelos experimentalistas. O critério adotado para a escolha destes projetos foi a disponibilidade de acesso ao código fonte e sua estrutura deveria permitir a avaliação de ambas as versões da ferramenta *Web Guitar*.

4.4. Execução do Estudo

Para execução do estudo, primeiramente foi necessário configurar o ambiente para execução das aplicações, conforme apresentado na Figura 47. Como pode ser observado, as aplicações Web ficaram hospedadas em um servidor Web local, enquanto que as duas versões da ferramenta *Web Guitar* foram disponibilizadas em outras duas máquinas.

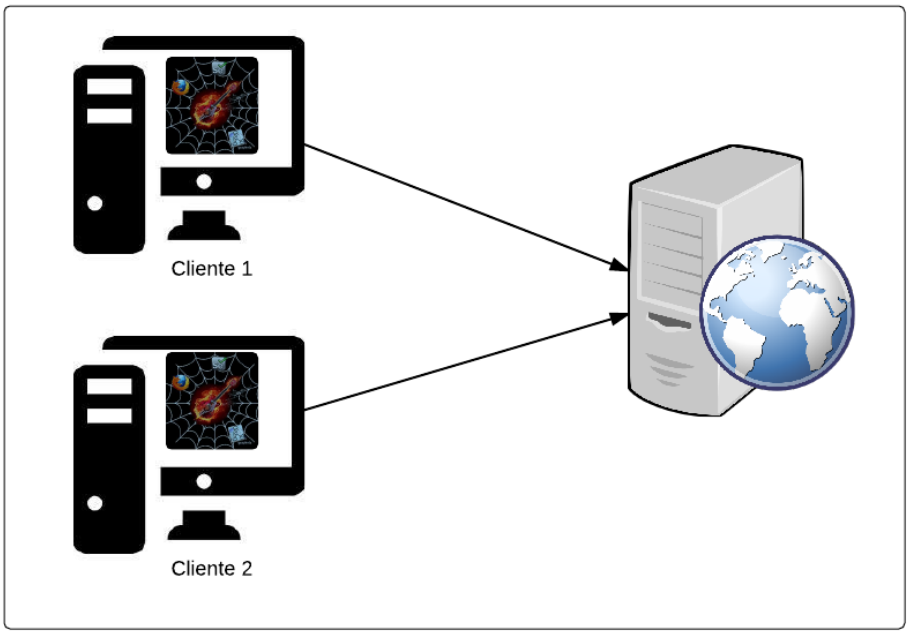

Figura 47: Ambiente de execução do experimento

As máquinas utilizadas deste experimento possuem as seguintes configurações:

- Servidor Web: Sistema Operacional Windows 8.1 de 64 bits, processador i7, 8 GB de memória RAM.
- Cliente 1: Sistema Operacional Linux distribuição Ubuntu 14.04 de 32 bits, processador i5 e 6 GB de memória RAM.
- Cliente 2: Sistema Operacional Linux distribuição Ubuntu 14.04 de 32 bits, processador i3 e 6 GB de memória RAM.

Neste estudo a máquina Cliente 1 executou os testes no Sistema de Apoio ao PPGI/UFAM, enquanto que o Cliente 2 executou os testes na aplicação Disk Transito. Além disso, durante a realização do experimento, as máquinas clientes permaneceram dedicadas a esta atividade.

O primeiro passo para realizar o experimento foi definir os parâmetros de entrada para execução do *Web Ripper*. Para cada aplicação, era necessário definir a URL base, o arquivo de configuração contendo as entradas manuais e a largura e profundidade desejada.

Como desejava-se saber o comportamento das duas versões da ferramenta *Web Guitar* com diferentes entradas, os valores de largura (w) e profundidade (d) foram previamente escolhidos seguindo a sequência de *Fibonacci*, pois a mesma representa uma escala exponencial que possibilitaria uma variação na execução dos algoritmos. Foi utilizada a sequência a partir do valor 3, até o valor 13, conforme a Figura 48.

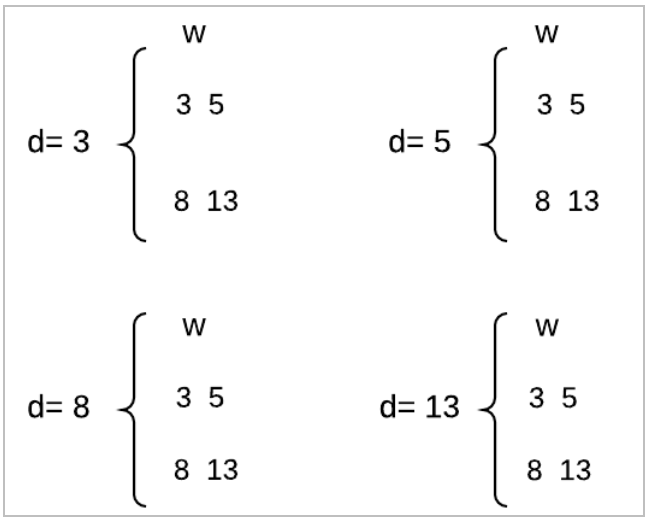

Figura 48: Definição dos valores de largura e profundidade

O valor da instância foi definido (i=2) para a ferramenta *WG-Modificada* e (i=1) para a ferramenta *WG-Original*, visto que a mesma explora apenas uma instância de cada página. Já o valor do número de visitas foi definido (v=3) apenas para a *WG-Modificada*, enquanto que para a *WG-Original* esse conceito não se aplica.

A URL base e as páginas ignoradas eram utilizadas para cada versão da aplicação testada para garantir que o ponto de partida e as páginas a serem ignoradas seriam as mesmas.

Os valores de entrada para a ferramenta *WG-Original* apenas continham os dados de *login* e senha, que permitiam que o *Web Ripper* acessasse a aplicação testada (caso contrário ela não passaria da tela inicial em ambos os sistemas). Já os dados de entrada para a ferramenta *WG-Modificada* eram os dados necessários para distinguir várias instâncias de uma página.

É importante ressaltar que o tempo necessário para definição dos parâmetros iniciais não foi contabilizado, como ocorreu em NGUYEN *et al.* (2013).

O processo de execução do módulo *Web Ripper* é apresentado na Figura 49. Uma vez definidos os parâmetros de entrada citados anteriormente, o *Web Ripper* era executado. Caso fosse observado que determinadas páginas demoravam a serem visitadas, a execução do *Web Ripper* seria interrompida e essas páginas seriam identificadas e incluídas na lista de página ignoradas do arquivo de configuração na próxima execução, exatamente como ocorreu no estudo original da ferramenta, publicado em (NGUYEN *et al.*, 2013). Neste estudo, não foram identificadas páginas ignoradas. Mesmo algumas páginas estáticas do site do Sistema de Apoio ao PPGI/UFAM foram exploradas.

Os dados de entradas para a ferramenta *WG-Modificada* eram editados a partir do arquivo de configuração de saída gerado por essa ferramenta.

A cada execução bem sucedida do *Web Ripper*, era anotado o tempo de execução e a quantidade de páginas exploradas para as duas versões da ferramenta.

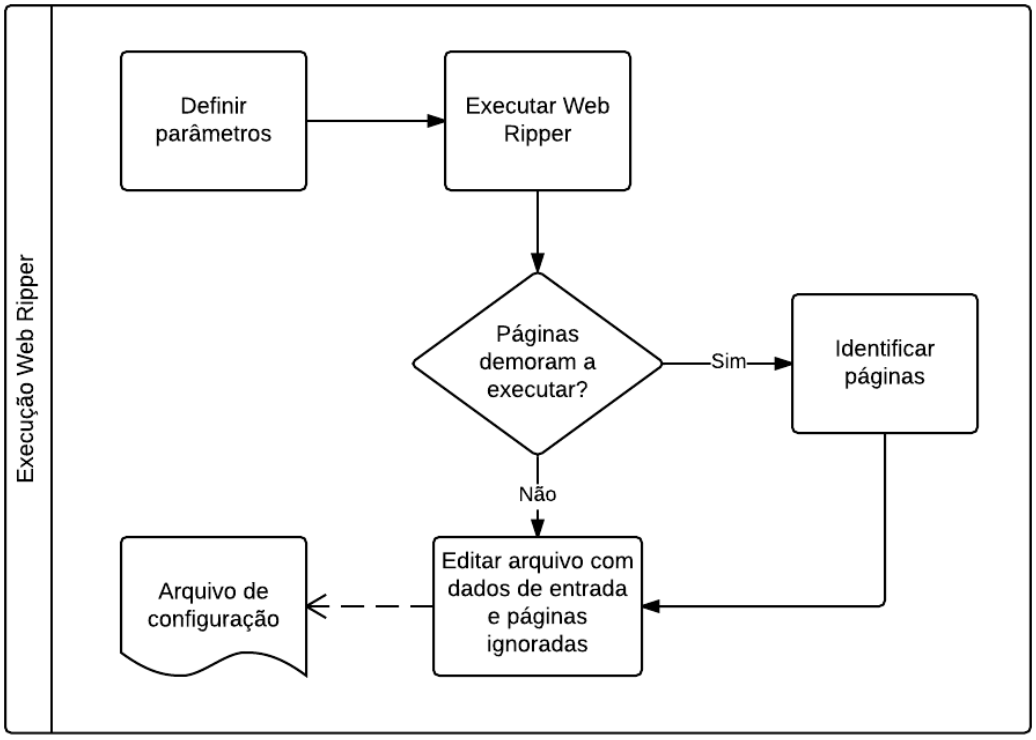

Figura 49: Execução do Web Ripper

O fluxo de execução do experimento é mostrado na Figura 50. Após execução do *Web Ripper* foram executados o Conversor EFG e Gerador de casos

de teste, os quais não sofreram nenhuma alteração em relação à *WG-Original* O Conversor de Grafo e Gerador de casos de teste não apresentavam variações no tempo de execução para as duas versões, por isso o tempo de execução dos mesmos não foi contabilizado.

Como o número de eventos que compõem um caso de teste influencia na quantidade de casos de teste gerados, uma vez que os eventos são combinados entre si, decidiu-se gerar apenas casos de teste de tamanho (l=2), ou seja, compostos por dois eventos, mesmo sabendo que essa decisão poderia influenciar na quantidade de caso de teste gerados.

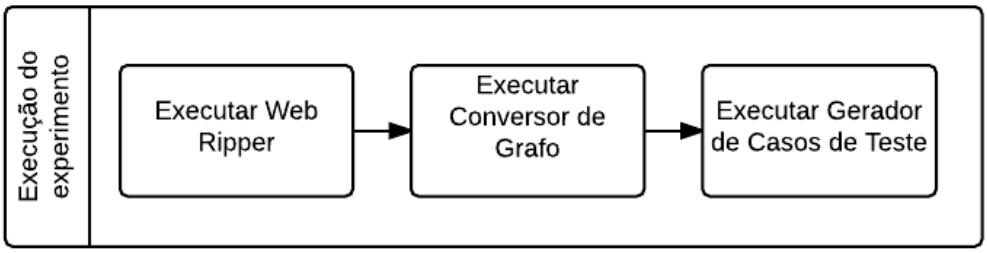

Figura 50: Execução do Experimento

É importante destacar que após a execução de cada rodada do experimento, a máquina cliente que o executava era reinicializada a fim de esvaziar sua memória RAM, evitando assim que possíveis lixos da memória afetassem o resultado da próxima execução.

4.5. Resultado do Estudo de Viabilidade

O resultado da execução do experimento é apresentado de acordo com os valores de profundidade definidos como parâmetro para execução do componente *Web Ripper*, variando-se os valores da largura para ambas as ferramentas. Para *WG-Modificada* manteve-se fixos os valores de instância da mesma página que pode ser visitada (i=2) e número máximo de tentativas de visitas (v=3). A quantidade de páginas é baseada em cada componente <*GUI*> da árvore GUI resultante. Portanto, para a *WG-Modificada*, a quantidade de páginas pode contabilizar a mesma URL no máximo duas vezes, desde que elas sejam instâncias diferentes da mesma página.

A seguir serão apresentados os resultados deste estudo, primeiro para o Sistema de Apoio ao PPGI/UFAM e depois para o Disk Transito.

4.5.1. Resultados Sistema de Apoio PPGI/UFAM

Na Tabela 6 são apresentados os resultados do estudo para o Sistema de Apoio PPGI/UFAM para profundidade (d=3). Pode-se observar que, no geral, a *WG-Modificada* explorou mais páginas e precisou de mais tempo para executar em relação a *WG-Original.*

Tabela 6: Resultado PPGI/UFAM para profundidade (d=3)

Ferramenta	Largura	Instância	Visitas	Tempo (s)	Páginas	CT	Tempo/CT
WG-Original	3	1	-	76	4	162	0,47
	5	1	-	113	6	1146	0,10
	8	1	-	156	6	1147	0,14
	13	1	-	125	6	1147	0,11
WG-Modificada	3	2	3	149	7	162	0,92
	5	2	3	183	11	162	1,13
	8	2	3	169	8	162	1,01
	13	2	3	170	8	162	1,05

De acordo com a Tabela 6, embora a *WG-Modificada* tenha explorado mais páginas em quase todas as execuções, a mesma não conseguiu gerar mais casos de teste, como demonstrado na Figura 51. Verificando-se que o número de casos de teste gerados manteve-se o mesmo para todas as larguras utilizadas e analisando-se a execução de ambas ferramentas, constatou-se que para a profundidade (d=3), o *Web Ripper* modificado encontrou duas instâncias de uma mesma página, porém quando o conversor de grafo gerava o EFG, ele somente utilizava a segunda. Esta é uma limitação observada em relação ao componente conversor de grafo, e foge ao escopo desta pesquisa, porém seria importante ser tratada em trabalhos futuros. Voltando à análise do estudo, a segunda instância era a mais profunda, a partir da qual não era possível acessar outras páginas, visto que se havia alcançado a profundidade máxima. Já para a largura (w=3), o número de casos de teste gerados foi similar, pois para esta configuração a *WG-Original* não conseguiu gerar mais casos de teste que a *WG-Modificada.*

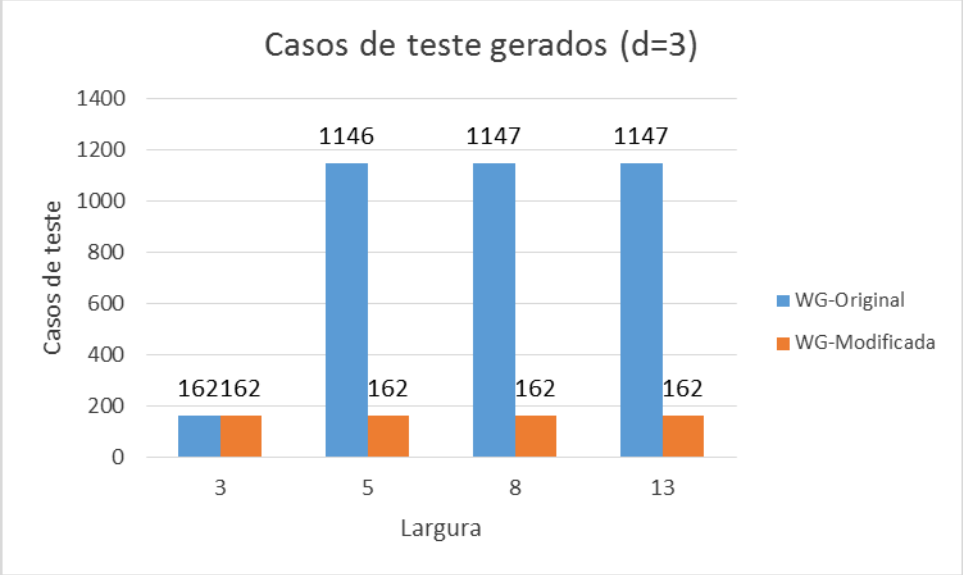

Figura 51: Gráfico de Casos de Teste gerados PPGI/UFAM (d=3)

O tempo de execução do *Web Ripper* foi dividido pelo número de casos de teste de tamanho (l=2), encontrando uma média do tempo utilizado para a geração de cada caso de teste. Na Figura 52, percebe-se que com a limitação observada no conversor de grafos, citada acima, a *WG-Modificada* utilizou mais tempo para visitar mais páginas, porém não conseguiu gerar mais casos de teste a partir do estado inicial e isso gerou grande diferença entre os valores da *WG-Original e WG-Modificada* para todas as larguras.

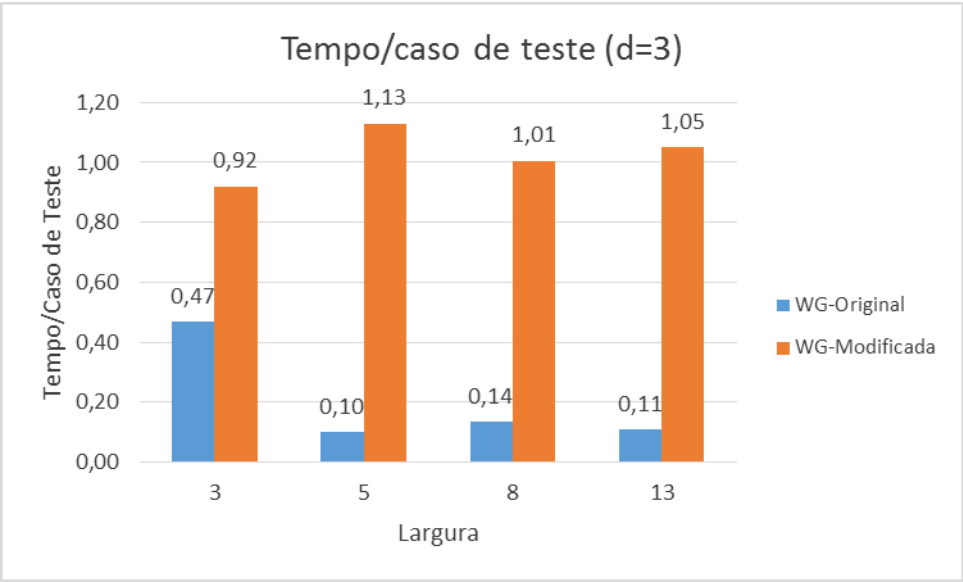

Figura 52: Gráfico de Tempo/Caso de Teste PPGI/UFAM (d=3)

Na Tabela 7 são apresentados os resultados do estudo para profundidade 5. Pode-se observar novamente que a *WG-Modificada* explorou mais páginas e precisou de mais tempo para executar que a *WG-Original*.

Tabela 7: Resultado PPGI/UFAM para profundidade (d=5)

Ferramenta	Largura	Instância	Visitas	Tempo (s)	Páginas	CT	Tempo/CT
WG-Original	3	1	-	129	6	50	2,58
	5	1	-	151	8	50	3,02
	8	1	-	153	8	50	3,06
	13	1	-	150	8	50	3,00
WG-Modificada	3	2	3	269	9	1156	0,23
	5	2	3	353	13	1189	0,30
	8	2	3	1192	16	1190	1,00
	13	2	3	1344	21	1190	1,13

Conforme pode-se observar na Figura 53, a *WG-Modificada* conseguiu gerar mais casos de teste, como consequência do fato de ter visitado mais páginas em todas as execuções e não ter apresentado o mesmo problema visto na configuração de profundidade (d=3). Para a profundidade (d=5), os casos de teste gerados pela *WG-Original* foram reduzidos significativamente, em consequência do tamanho dos casos de teste (l=2) utilizados e da mesma não visitar novamente uma página já visitada.

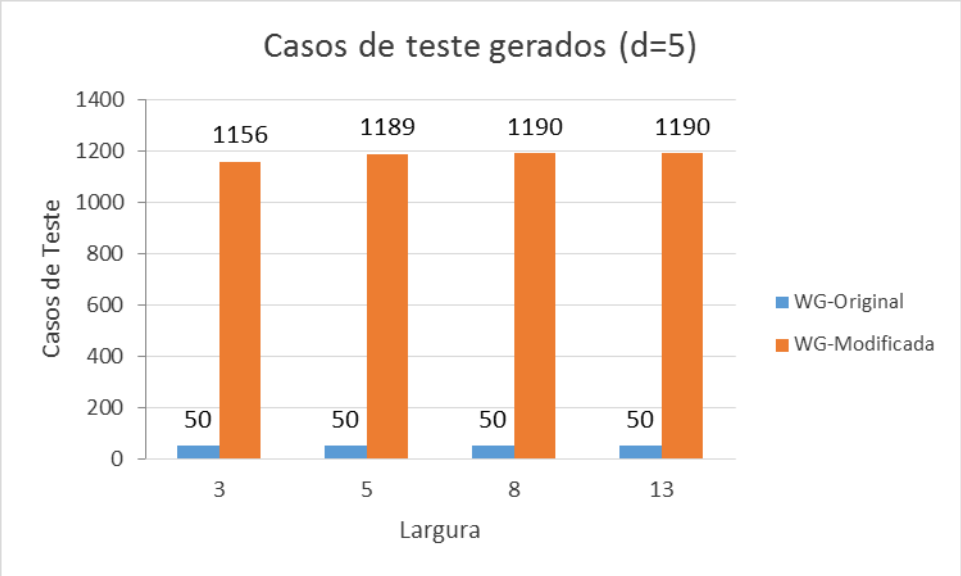

Figura 53: Gráfico de Casos de Teste gerados PPGI/UFAM (d=5)

Em relação ao tempo necessário para encontrar cada caso de teste de tamanho 2 (Figura 54), verificou-se que o tempo a mais de execução da *WG-Modificada* foi compensado pela grande quantidade de casos de teste gerados. Esse

fato ocasionou uma grande diferença entre os valores da *WG-Original e WG-Modificada*.

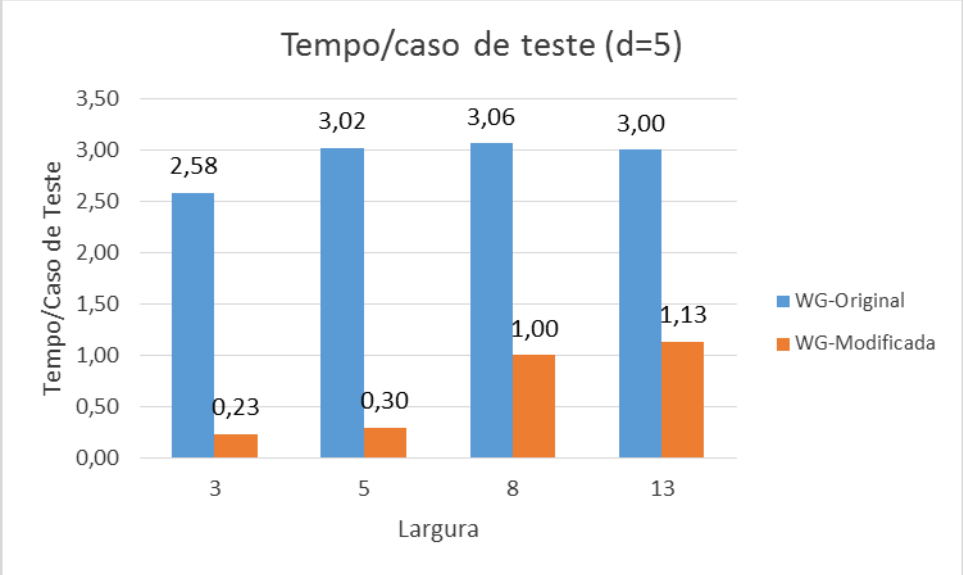

Figura 54: Gráfico de Tempo/Caso de Teste PPGI/UFAM (d=5)

Na Tabela 8 estão os resultados do estudo para profundidade (d=8). Pode-se observar novamente que a *WG-Modificada* encontrou uma quantidade de páginas maior que a *WG-Original.*

Tabela 8: Resultado PPGI/UFAM para profundidade (d=8)

Ferramenta	Largura	Instância	Visitas	Tempo (s)	Páginas	CT	Tempo/CT
WG-Original	3	1	-	178	7	50	3,56
	5	1	-	285	7	50	5,70
	8	1	-	1179	14	50	23,58
	13	1	-	1458	14	50	29,16
WG-Modificada	3	2	3	276	11	50	5,52
	5	2	3	380	14	68	5,59
	8	2	3	513	14	68	7,54
	13	2	3	511	14	68	7,51

Conforme pode-se observar na Figura 55, novamente a *WG-Modificada*, por ter visitado mais páginas, conseguiu gerar mais casos de teste para todas as larguras, com exceção da largura (w=3), na qual visitou-se mais páginas, mas isso não foi suficiente para que fossem gerados mais casos de teste.

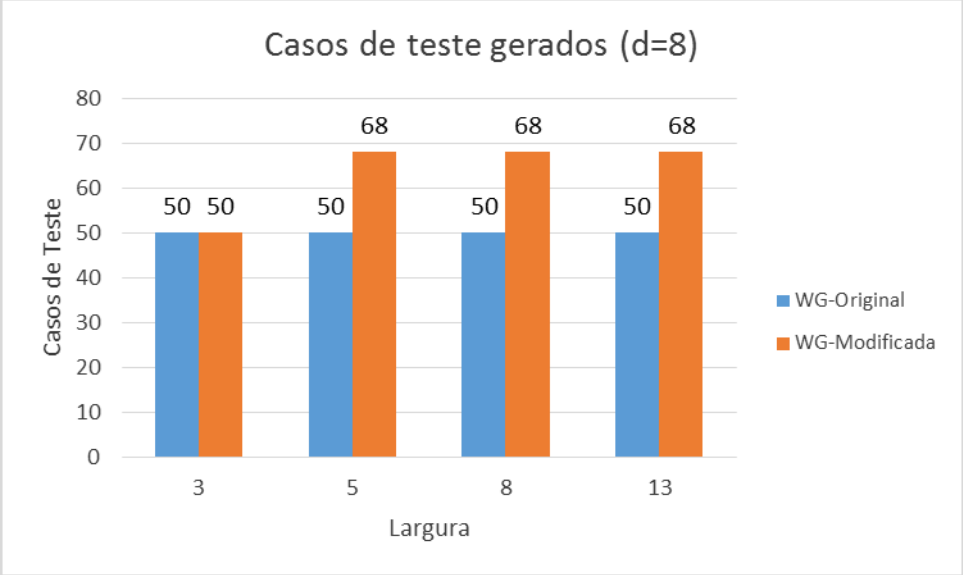

Figura 55: Gráfico de casos de Teste gerados PPGI/UFAM (d=8)

É possível perceber que para as larguras 8 e 13, a *WG-Original* demorou mais tempo na sua execução. Foi constatado que essa ferramenta demorou tempo excessivo para percorrer a página mostrada na Figura 56, em que cada componente de entrada dentro do elemento destacado possui uma função *javascript* associada.

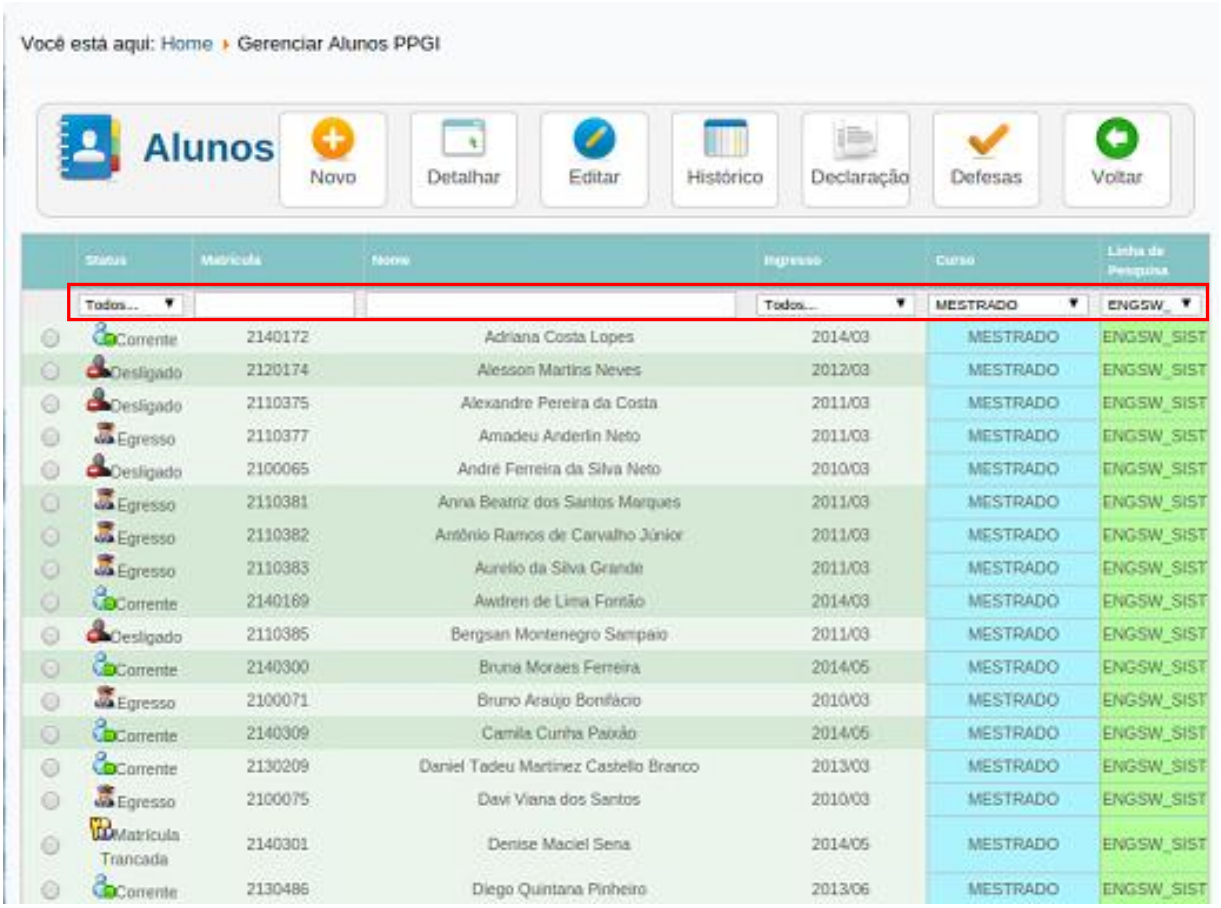

Figura 56: Página Gerenciar Alunos PPGI

Em relação ao tempo necessário para encontrar cada caso de teste (Figura 57), nota-se que a *WG-Modificada* gerou os casos de teste com tempo inferior para as larguras 5, 8 e 13, apesar de que o tempo de execução do *Web Ripper* ter sido maior.

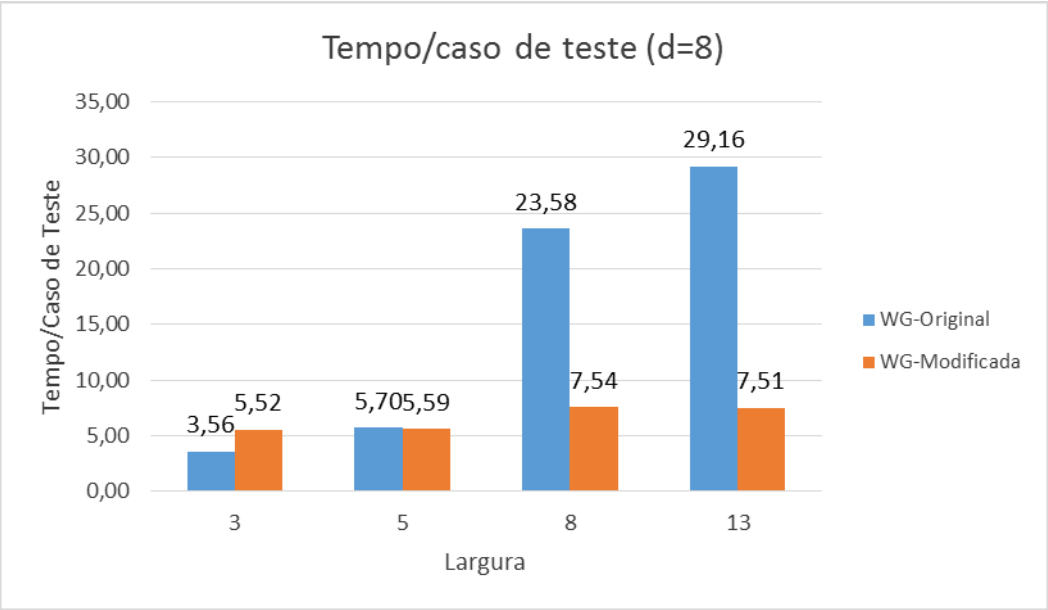

Figura 57: Gráfico de Tempo/Caso de Teste PPGI/UFAM (d=8)

Na Tabela 9 estão os resultados do estudo para profundidade (d=13). Observa-se que novamente a *WG-Modificada* encontrou uma quantidade de páginas maior, porém precisou de mais tempo para executar que a *WG-Original.*

Tabela 9: Resultado PPGI/UFAM para profundidade (d=13)

Ferramenta	Largura	Instância	Visitas	Tempo (s)	Páginas	CT	Tempo/CT
WG-Original	3	1	-	189	7	50	3,78
	5	1	-	300	11	50	6,00
	8	1	-	1278	14	50	25,56
	13	1	-	1556	19	50	31,12
WG-Modificada	3	2	3	427	11	50	8,54
	5	2	3	800	15	1099	0,73
	8	2	3	3471	19	50	69,42
	13	2	3	3989	24	50	79,78

Conforme pode-se observar na Figura 58, a quantidade de casos de teste gerados é similar para as duas ferramentas, com exceção da configuração de largura (w=5), em que a *WG-Modificada* percorreu um caminho que lhe permitiu gerar mais casos de teste a partir da URL base.

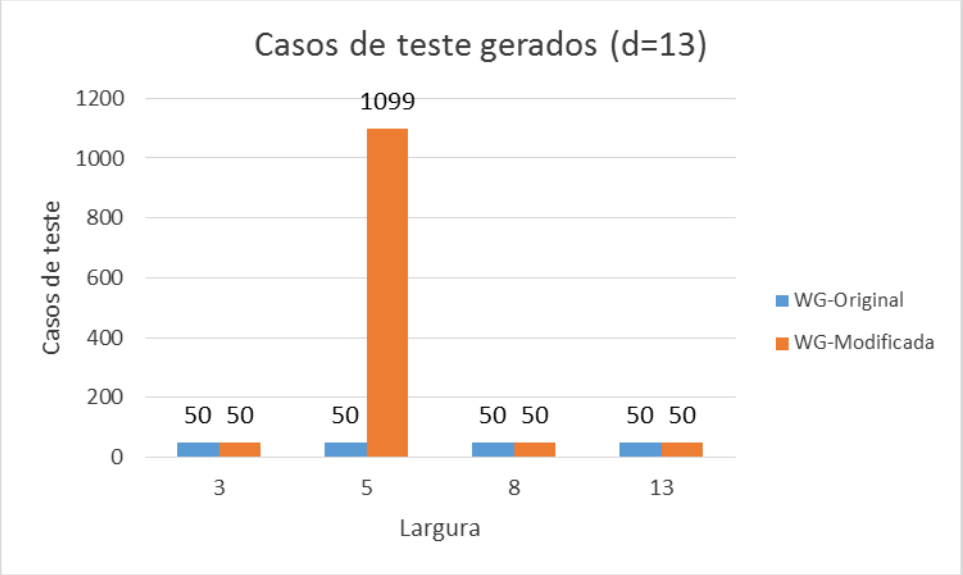

Figura 58: Gráfico de casos de teste gerados PPGI/UFAM (d=13)

Em relação ao tempo necessário para encontrar cada caso de teste (Figura 59), nota-se que a *WG-Original* apresentou melhor relação de tempo por casos de teste. Apenas para a largura (w=5) a *WG-Modificada* conseguiu reduzir o tempo necessário para gerar os casos de testes

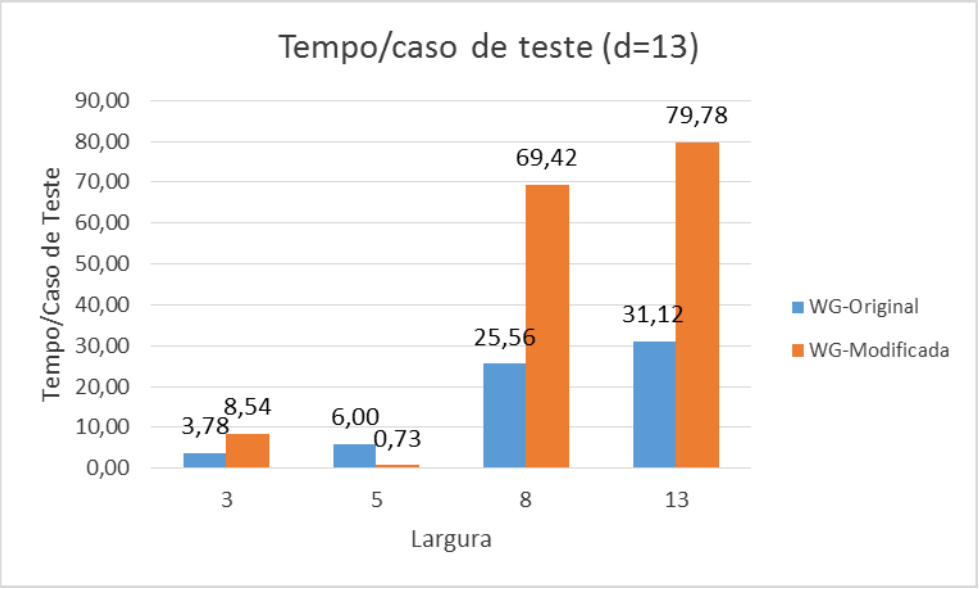

Figura 59: Gráfico de Tempo/Caso de Teste PPGI/UFAM (d=13)

A partir dos resultados das execuções do experimento para o projeto PPGI/UFAM, foi elaborada a Tabela 10, na qual cada linha representa uma configuração, ou seja, a combinação de profundidade e largura. Essa tabela contém as diferenças de valores entre a *WG-Modificada* e a *WG-Original* (*WG-Modificada* –

WG-Original). Assim, uma célula com valor positivo (>0) indica que *WG-Modificada* apresentou um valor maior para o atributo quando comparada à versão *WG-Original*. Se WG-Original obteve valor maior que WG-Modificada, então o valor da célula será negativo. Caso os valores sejam iguais, a célula terá o valor 0 (zero). As células destacadas são aquelas em que *WG-Modificada* apresentou um melhor resultado em comparação com a versão *WG-Original*.

Tabela 10: Resultados do Sistema de Apoio ao PPGI/UFAM

Profundidade	Largura (w)	CT	Páginas	Tempo (s)	Tempo/CT
3	3	0	3	-73	0,45
	5	-984	5	-70	1,03
	8	-985	2	-13	0,87
	13	-985	2	-45	0,94
5	3	1106	3	-140	-2,35
	5	1139	5	-201	-2,72
	8	1140	8	-1039	-2,06
	13	1140	12	-1194	-1,87
8	3	0	4	-98	1,96
	5	18	7	-95	-0,11
	8	18	0	666	-16,04
	13	18	0	947	-21,65
13	3	0	4	-238	4,76
	5	1049	1	-500	-5,27
	8	0	5	-2193	43,86
	13	0	6	-2433	48,66

Analisando-se a variável *CT*, é possível notar que em 8 (50,00%) das configurações a *WG-Modificada* gerou um número dos casos de teste superior a *WG-Original.* Em 5 (31,25%) das configurações não houve variação no número dos casos de teste gerados entre as versões*.* Apenas para profundidade (d=3), a *WG-Modificada* apresentou valores inferiores a *WG-Original* (3: 18,75%)*,* isso porque para essa profundidade foi evidenciado o problema de identificação de instâncias relacionado ao módulo Conversor de Grafo EFG, o qual não foi alterado neste trabalho.

Analisando-se a variável *Páginas*, é possível observar que em 14 (87,50%) das configurações a *WG-Modificada* encontrou um número superior de páginas. Em 2 (12,50%) das configurações não houve variação no número de páginas. É importante destacar nessas configurações que embora tenha sido visitado o mesmo

número de páginas, as páginas visitadas não foram as mesmas, por isso as duas ferramentas geraram um número de casos de teste diferente.

Analisando-se a variável *Tempo*, é possível verificar que em 14 (87,50%) das configurações a *WG-Modificada* necessitou de um tempo superior para executar, em consequência do fato que a mesma necessitar revisitar páginas já visitadas ao menos uma vez para verificar se a mesma distingue-se da instância da página já visitada. Em 2 (12,50%) das configurações a *WG-Original* necessitou de mais tempo para executar. Essa seria uma variável que requer melhorias e novas avaliações no futuro.

Analisando-se a variável *Tempo/CT*, é possível notar que em 8 (50,00%) das configurações a *WG-Modificada* foi mais eficiente em gerar casos de teste, visto que o tempo maior de execução do *Web Ripper* foi compensado pelo número de casos de testes gerados. Na outra metade das configurações (8: 50,00%) a *WG-Original* foi mais eficiente em gerar casos de teste *que* a *WG-Modificada*, resultado dos cenários já relatados anteriormente.

Alguns testes foram realizados novamente com os mesmos parâmetros de profundidade, largura, instâncias[6], visitas[4] *e* arquivo de configuração e foram obtidos os mesmos resultados de número de páginas e CT gerados, houve porém uma variação do tempo em alguns segundos e, consequentemente, variando-se a variável Tempo/CT.

Fatores que podem ocasionar um resultado diferente do resultado apresentado são:

- Alteração no arquivo de configuração, cujos dados permitem distinguir várias instâncias de uma mesma página;
- Problema relacionados a infraestrutura, tais como velocidade da conexão e características das máquinas utilizadas (Clientes e servidor).

4.5.2. Resultados Disk Transito

Na Tabela 11 são apresentados os resultados do estudo para o sistema Disk Transito para profundidade (d=3). Pode-se observar que a *WG-Modificada* explorou mais páginas e precisou mais tempo para executar que a *WG-Original.*

[6] Utilizado somente na *WG-Modificada.*

Tabela 11: Resultado Disk Transito para profundidade (d=3)

Ferramenta	Largura	Instância	Visitas	Tempo (s)	Páginas	CT	Tempo/CT
WG-Original	3	1	-	110	4	780	0,14
	5	1	-	277	6	1734	0,16
	8	1	-	467	10	3313	0,14
	13	1	-	612	12	3673	0,17
WG-Modificada	3	2	3	127	5	780	0,16
	5	2	3	469	8	2184	0,21
	8	2	3	1168	12	4380	0,27
	13	2	3	655	19	4516	0,15

Diante da adição de novos recursos, a *WG-Modificada* explorou mais páginas em todas as execuções, e conseguiu gerar o mesmo número de casos de teste em um caso largura (w=3) e mais casos de teste nos demais, conforme pode ser observado na Figura 60.

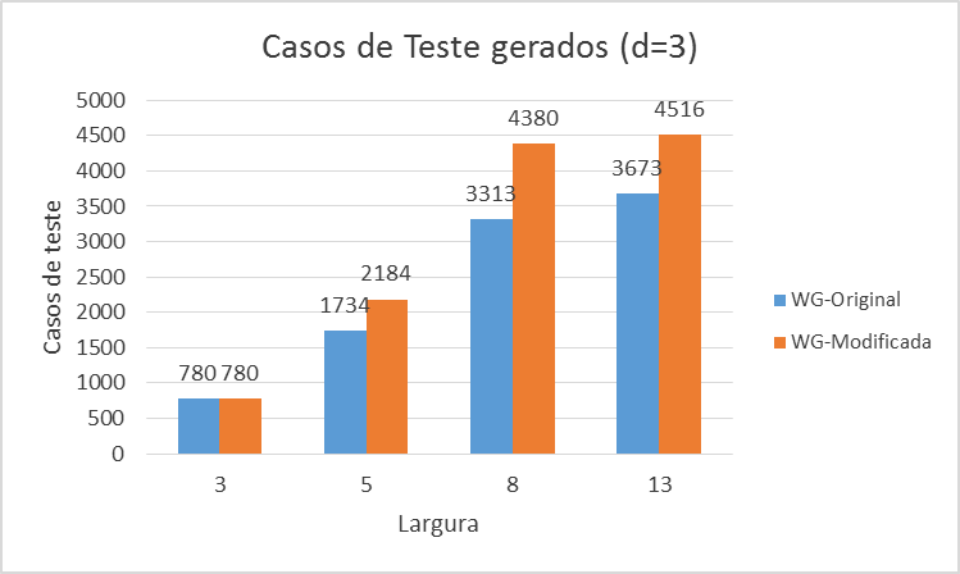

Figura 60: Gráfico de casos de teste gerados Disk Transito (d=3)

Em relação ao tempo necessário para encontrar cada caso de teste (Figura 61), pode-se perceber que mesmo a *WG-Modificada* tendo gerado mais casos de teste, o tempo gasto para sua execução foi superior ao tempo de execução da *WG-Original,* com exceção da configuração de largura (w=13), cuja relação tempo/caso de teste é menor que a *WG-Original.*

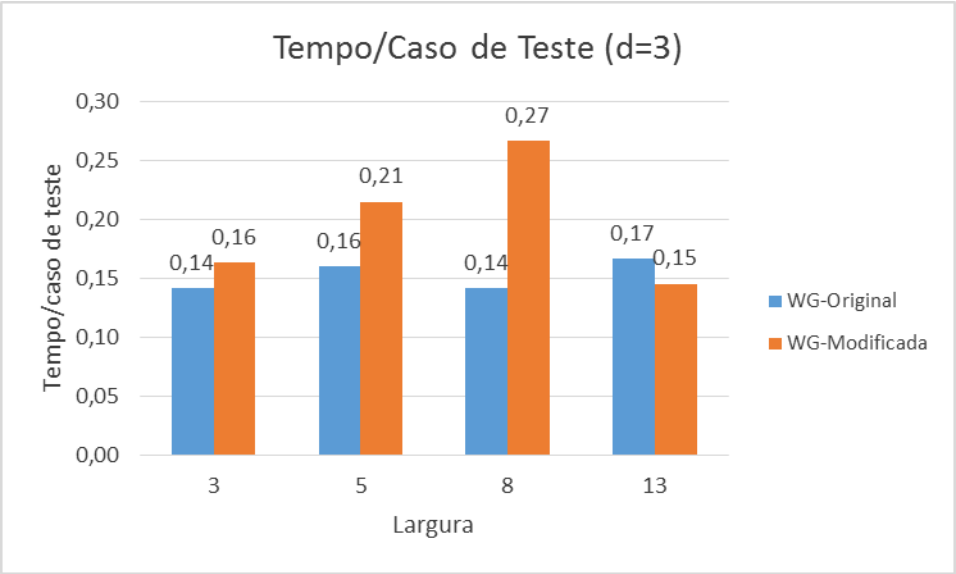

Figura 61: Gráfico de Tempo/Caso de Teste Disk Transito (d=3)

Na Tabela 12 são apresentados os resultados do estudo para profundidade (d=5). Pode-se observar que a *WG-Modificada* explorou mais páginas e necessitou um tempo maior para executar que a *WG-Original.*

Tabela 12: Resultado Disk Transito para profundidade (d=5)

Ferramenta	Largura	Instância	Visitas	Tempo (s)	Páginas	CT	Tempo/CT
WG-Original	3	1	-	86	4	73	1,18
	5	1	-	363	6	272	1,33
	8	1	-	506	10	3227	0,16
	13	1	-	826	17	6050	0,14
WG-Modificada	3	2	3	202	7	143	1,41
	5	2	3	864	14	321	2,69
	8	2	3	1117	17	321	3,48
	13	2	3	1184	33	271	4,37

À medida que a *WG-Modificada* explorou mais páginas em todas as execuções, a mesma conseguiu gerar mais casos de teste para as configurações de menor largura (3 e 5), porém para as larguras 8 e 13 o número de casos de teste gerados pela *WG-Original* foram superiores a *WG-Modificada*, conforme pode-se observar na Figura 62.

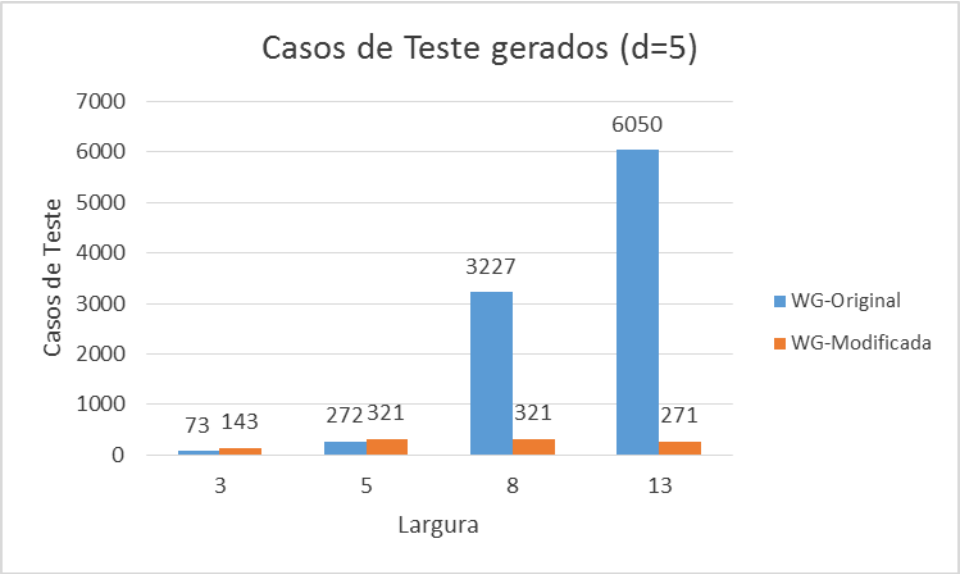

Figura 62: Gráfico de casos de teste gerados Disk Transito (d=5)

Para essa profundidade, ao aumentar-se a largura, ambas as ferramentas exploram a página da Figura 63, que possui vários botões *javascript*, os quais não geram eventos e que possuem uma única instância. Para a *WG-Original*, o *Web Ripper* tenta visitar cada página referenciada em cada botão uma única vez, já para a *WG-Modificada* isso é feito três vezes, uma vez que o número de visitas foi definido em (v=3), tornando a execução da *WG-Modificada* mais demorada. A partir desta profundidade, observa-se também que o problema de identificação de instâncias do Conversor de grafo EFG, descrito anteriormente, torna-se evidente, visto que aumenta-se o número de páginas percorridas, porém não se aumenta o número de casos de teste gerados.

Um outro problema observado a partir dessa profundidade é a questão de preenchimento de dados de entrada, que torna a execução da *WG-Modificada* mais lenta, à medida que aumenta-se a quantidade de elementos de entrada de uma página.

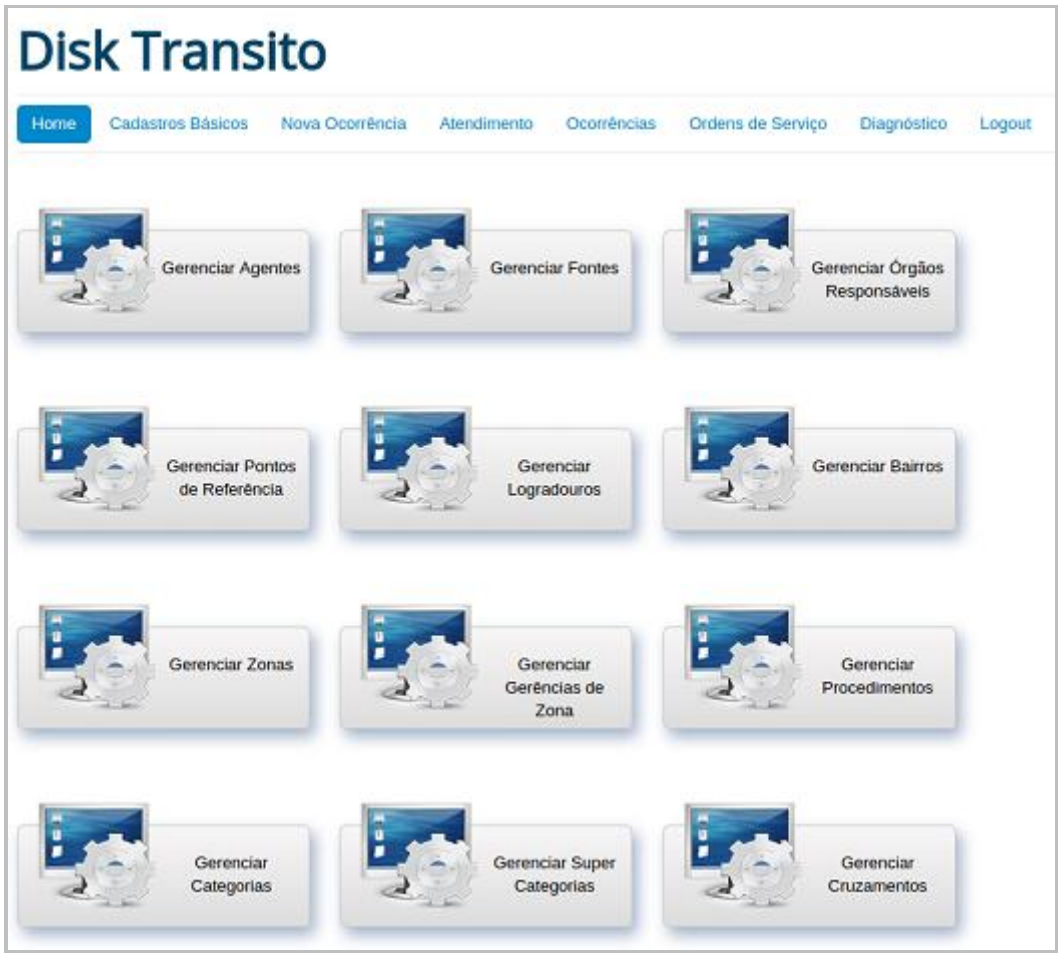

Figura 63: Página de Cadastros básicos

Em relação ao tempo necessário para encontrar cada caso de teste (Figura 64), percebe-se que para todas as larguras o tempo gasto para executar a *WG-Modificada* foi superior que o tempo de execução da *WG-Original.*

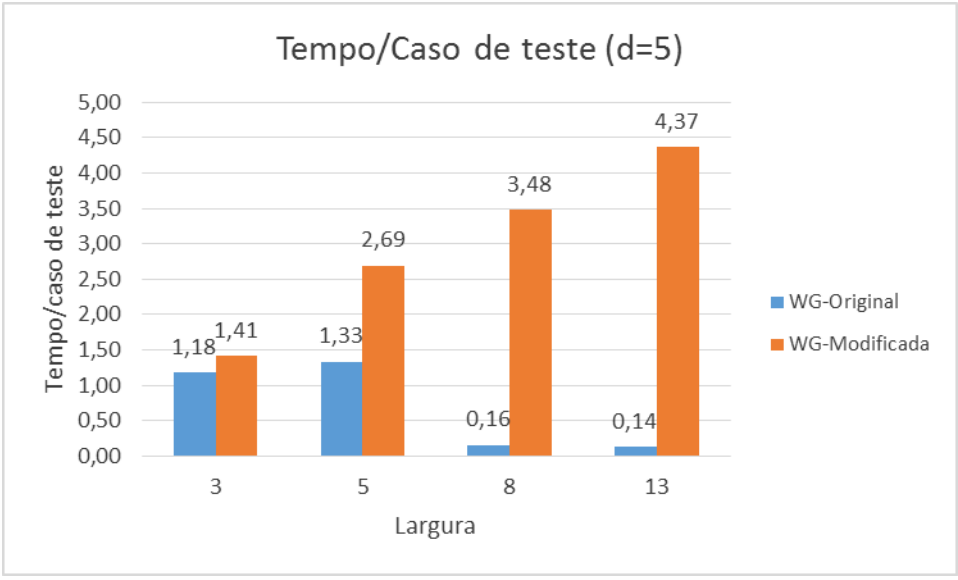

Figura 64: Gráfico de Tempo/Caso de Teste Disk Transito (d=5)

Na Tabela 13 são apresentados os resultados do estudo para profundidade (d=8). Pode-se observar que a *WG-Modificada* explorou mais páginas e necessitou de um tempo superior para ser executada, quando comparada à *WG-Original.*

Tabela 13: Resultado Disk Transito para profundidade (d=8)

Ferramenta	Largura	Instância	Visitas	Tempo (s)	Páginas	CT	Tempo/CT
WG-Original	3	1	-	119	4	750	0,16
	5	1	-	234	6	272	1,19
	8	1	-	580	10	3278	0,18
	13	1	-	1295	39	204	6,35
WG-Modificada	3	2	3	220	7	143	1,54
	5	2	3	753	12	67	11,24
	8	2	3	1206	18	67	18,00
	13	2	3	1455	39	271	5,37

A *WG-Modificada* embora tenha explorado mais páginas em todas as execuções, não conseguiu gerar mais casos de teste que *WG-Original*, conforme Figura 65. Isso é ocasionado pelos problemas descritos na profundidade (d=5), identificação de instâncias e páginas que possuem grande quantidade de botões *javascript.*

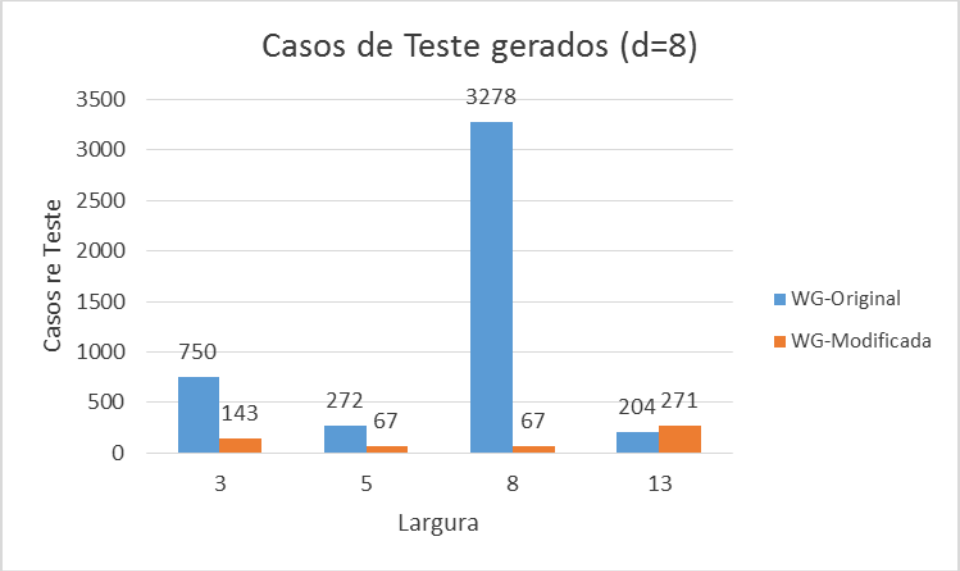

Figura 65: Gráfico de casos de teste gerados Disk Transito (d=8)

Em relação ao tempo necessário para encontrar cada caso de teste (Figura 66), pode-se perceber que para as larguras (3, 5 e 8) o tempo gasto para executar a *WG-Modificada* foi superior que o tempo de execução da *WG-Original.* Para a largura 13*, a WG-Modificada* encontra um caminho diferente do caminho encontrado nas larguras anteriores.

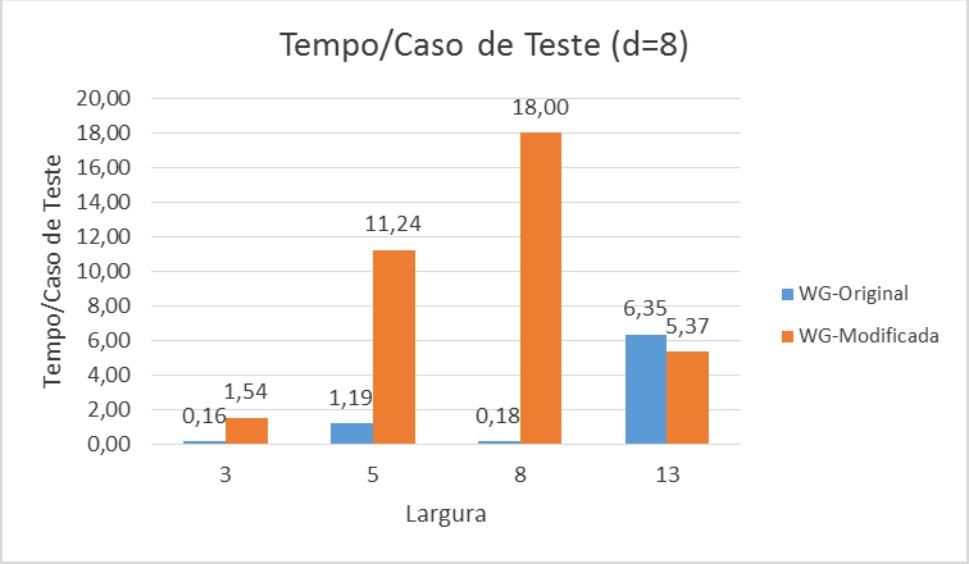

Figura 66: Gráfico de Tempo/Caso de Teste Disk Transito (d=8)

Na Tabela 14 são apresentados os resultados do estudo para profundidade 13. Pode-se observar que a *WG-Modificada* explorou mais páginas e necessitou um tempo superior para executar que a *WG-Original.*

Tabela 14: Resultado Disk Transito para profundidade (d=13)

Ferramenta	Largura	Instância	Visitas	Tempo (s)	Páginas	CT	Tempo/CT
WG-Original	3	1	-	119	4	750	0,16
	5	1	-	387	6	272	1,42
	8	1	-	672	10	3151	0,21
	13	1	-	2411	46	204	11,82
WG-Modificada	3	2	3	218	7	143	1,52
	5	2	3	844	12	67	12,60
	8	2	3	1357	15	67	20,25
	13	2	3	2474	53	271	9,13

A WG-Modificada embora tenha explorado mais páginas em todas as execuções, não conseguiu gerar mais casos de teste que a *WG-Original*, conforme a Figura 67. Para largura (w=8), pode-se verificar que a *WG-Original* conseguiu gerar um elevado número de casos de teste, isso em consequência de ter alcançado um caminho que lhe permitiu gerar uma grande quantidade de eventos a partir da URL base.

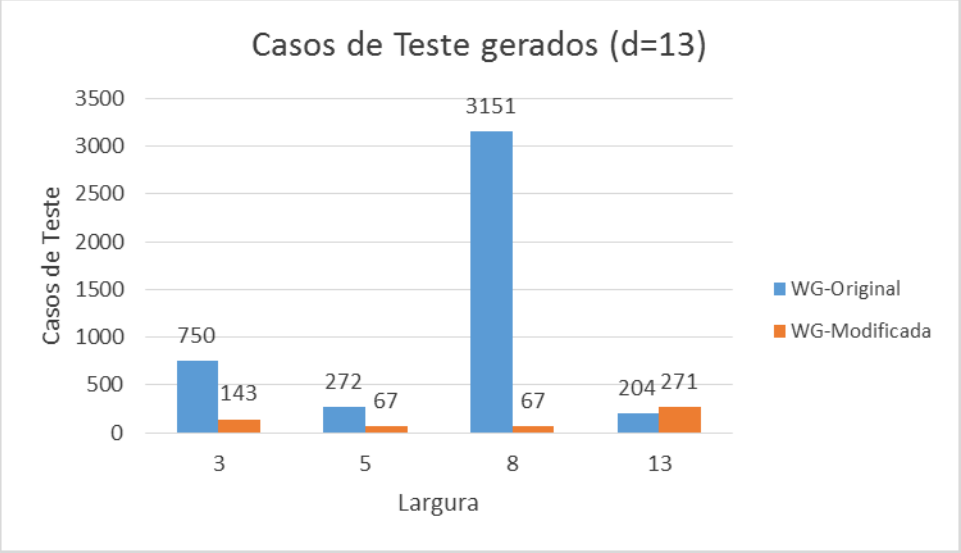

Figura 67: Gráfico de casos de teste gerados Disk Transito (d=13)

A *WG-Original*, embora tenha explorado uma quantidade menor de páginas em todas as execuções, conseguiu gerar mais casos de teste que a *WG-Modificada*, conforme a Figura 68. Novamente para a largura (w=13)*, a WG-Modificada* encontra um caminho diferente do caminho encontrado nas larguras anteriores.

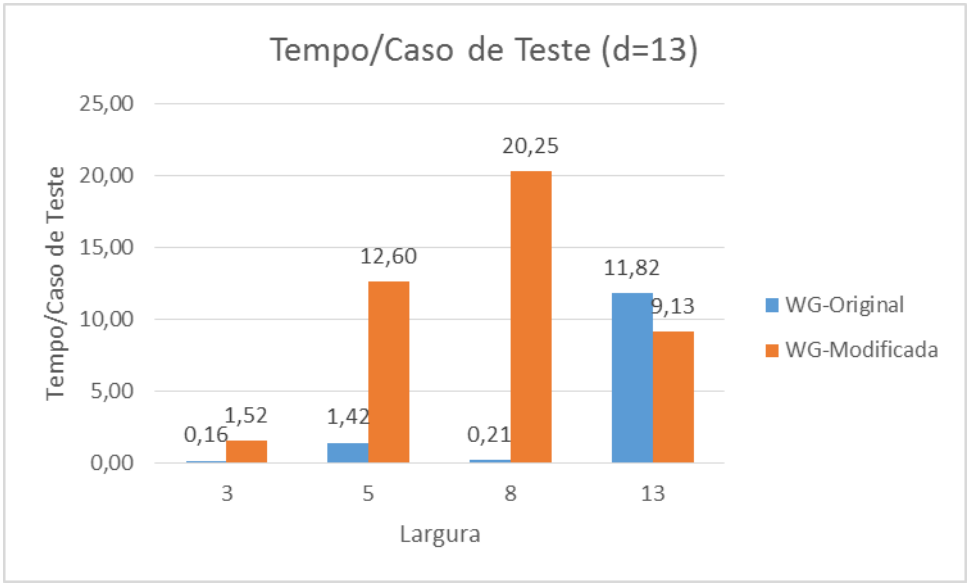

Figura 68: Gráfico de Tempo/Caso de Teste Disk Transito (d=13)

A partir dos resultados das execuções do experimento para o projeto Disk Transito, foi elaborada a Tabela 15, assim como foi feito para o projeto anterior, na qual cada linha representa uma configuração, ou seja, a combinação de profundidade e largura. Essa tabela contém as diferenças de valores entre a *WG-Modificada* e a *WG-Original* (*WG-Modificada – WG-Original*). Assim, novamente, uma célula com valor positivo (>0) indica que WG-Modificada apresentou um valor

maior para o atributo quando comparada à versão *WG-Original*. Se WG-Original obteve valor maior que WG-Modificada, então o valor da célula será negativo. Caso os valores sejam iguais, a célula terá o valor 0 (zero). As células destacadas são aquelas em que *WG-Modificada* apresentou um melhor resultado em comparação com a versão *WG-Original*.

Analisando-se a variável *CT* é possível notar que em 7 (43,75%) das configurações, a *WG-Modificada* gerou um número dos casos de teste superior a *WG-Original.* Em (1: 6,25%) das configurações não houve variação no número dos casos de teste gerados entre as versões*.* Já em 8 (50,00%) das configurações a *WG-Original* gerou um número dos casos de teste superior a *WG-Modificada.*

Analisando-se a variável *Páginas*, é possível observar que em 15 (93,75%) das configurações a *WG-Modificada* encontrou um número superior de páginas. Apenas em 1 (6,25%) das configurações não houve variação no número de páginas.

Analisando-se a variável *Tempo*, é possível verificar que em todas (100,00%) as configurações a *WG-Modificada* necessitaram de um tempo superior para executar, em consequência do fato que a mesma necessita revisitar uma página ao menos mais uma vez (no máximo três) para verificar se a mesma distingue-se da instância da página já visitada, conforme definido anteriormente.

Tabela 15: Resultados do Disk Transito

Profundidade	Largura (w)	CT	Páginas	Tempo(s)	Tempo/CT
3	3	0	1	7	0,02
	5	450	2	192	0,05
	8	1067	2	701	0,13
	13	843	7	43	-0,02
5	3	70	3	116	0,23
	5	49	8	501	1,36
	8	-2906	7	611	3,32
	13	-5779	16	358	4,23
8	3	-607	3	101	1,38
	5	-205	6	519	10,05
	8	-3211	8	626	17,82
	13	67	0	160	-0,98
13	3	-607	3	99	1,37
	5	-205	6	457	11,17
	8	-3084	5	685	20,04
	13	67	7	63	-2,69

Analisando-se a variável *Tempo/CT*, é possível notar que em (13: 81,25%) das configurações a *WG-Original* foi mais eficiente em gerar casos de teste. Apenas em (3: 18,75%) das configurações a *WG-Modificada* foi mais eficiente em gerar casos de teste *que* a *WG-Original.*

Caso este estudo seja repetido com as mesmas configurações de profundidade, largura, instâncias[7], visitas[6] *e* arquivo de configuração, esperam-se encontrar os mesmos resultados de número de páginas e CT gerados, podendo-se variar o tempo em alguns segundos e, consequentemente, o Tempo/CT.

Fatores que podem ocasionar um resultado diferente do resultado apresentado são:

- Alteração no arquivo de configuração, cujos dados permitem distinguir várias instâncias de uma mesma página;
- Problema relacionados a infraestrutura, tais como velocidade da conexão e características das máquinas utilizadas (clientes e servidor).

4.6. Ameaças à Validade do Estudo

Nesta seção serão detalhadas as possíveis ameaças à validade dos resultados.

4.6.1. Validade Interna

Ameaças à validade interna podem indicar uma relação causal inexistente entre tratamento e resultado. As principais ameaças identificadas foram:

- **Instrumentação:** Foi realizada a alteração no código fonte da *WG-Original* para incluir suporte a dados de entrada, pois não era possível enviar os dados necessários e as aplicações testadas exigiam registrar-se para a acessar suas funcionalidades. Como este é um estudo de viabilidade, acredita-se que essa questão não representa uma ameaça significativa.
- **Configuração:** Uma vez que configuração da ferramenta em ambas as versões (*WG-Original* e *WG-Modificada*) pode influenciar na sua execução, é necessário reduzir os riscos causados pela mesma. Por isso a configuração dos parâmetros foi realizada conforme orientações dos autores, descritas em NGUYEN *et al.* (2013).

[7] Utilizado somente na *WG-Modificada.*

- **Seleção dos estudos:** Amostra dos projetos não foi aleatória, visto que era proveniente da disponibilidade de projetos dentro do grupo de pesquisa no qual o trabalho foi realizado. Como este é um estudo de viabilidade, acredita-se que essa questão não representa uma ameaça significativa.

4.6.2. Validade Externa

Ameaças à validade externa estão relacionadas com capacidade de generalizar resultados do experimento fora do seu ambiente. As principais ameaças identificadas foram:

- **Interação entre Ambiente e Tratamento:** Para reduzir os efeitos causados pela redução da velocidade de conexão, ao perceber que o carregamento de páginas estava demorando, a execução era parada e reiniciada quando normalizada a velocidade da conexão. Isso era feito para cada configuração em ambas as versões da ferramenta.
- **Generalização dos Resultados:** Como neste trabalho foi utilizado um número limitado de aplicações não é possível generalizar os resultados. Futuramente, pretende-se aumentar o número e o tipo de aplicações Web.

4.6.3. Validade de Construção

Ameaças à validade de construção referem-se à extensão que a configuração do experimento reflete na construção em estudo. A ameaça identificada foi:

- **Viés mono-operação:** o estudo de viabilidade utilizou dois projetos Web desenvolvidos em linguagem PHP com o *framework Joomla.* O estudo pode não ser representativo para outros *frameworks* ou outras linguagens.

4.6.4. Validade de Conclusão

Ameaças à validade de conclusão estão preocupadas com questões que afetam a capacidade de tirar a conclusão correta sobre as relações entre o tratamento e o resultado.

- **Confiança das Medidas:** Percebendo-se que resultados de execuções anteriores poderiam impactar nas próximas execuções, a cada execução do *Web Ripper* (*WG-Original e WG-Modificada*), a máquina cliente era reiniciada. Outra questão estava relacionada as configurações em que a diferença de

resultados foi significativa. Para esses casos, os testes foram refeitos para garantir que o tempo, CT e páginas eram as mesmas.

4.7. Considerações Finais

Neste capítulo, foi realizado um estudo da aplicação da ferramenta *WG-Modificada* para avaliar sua viabilidade no que diz respeito a geração de casos de teste em aplicações Web reais.

Como as aplicações testadas representam apenas a categoria transacional, os resultados alcançados não podem ser considerados conclusivos. No entanto, o resultado da avaliação geral foi considerado positivo. A análise do experimento apresentou indícios da viabilidade da utilização da ferramenta uma vez que ela conseguiu gerar mais casos de teste para o Sistema de Apoio ao PPGI/UFAM, porém os resultados do estudo para o Disk Transito se apresentaram melhores para a ferramenta *WG-Original,* o que evidenciou problemas e limitações na *WG-Modificada* que necessitam ser resolvidos, como o problema de identificação de instâncias do Conversor EFG e também problemas relacionados ao preenchimento de dados de entrada que aumentam o tempo de execução.

Como limitação deste estudo, há a necessidade de completar o processo de testes da ferramenta *Web Guitar*, seguindo com a execução do *Web Replayer* para que os casos de teste gerados por ambas as ferramentas, e analisando a taxa de detecção de defeitos dos novos casos de teste gerados pela *WG-Modificada*.

No próximo capítulo serão apresentadas as contribuições decorrentes deste trabalho, assim como limitações encontradas e trabalhos futuros que já estão sendo planejados.

CAPÍTULO 5 – CONCLUSÃO

Neste capítulo serão apresentadas as conclusões desta dissertação, ressaltando suas contribuições e trabalhos futuros que fornecem a direção para que seja dada continuidade a esta pesquisa, além de suas limitações.

5.1. Considerações Finais

Este trabalho apresentou a evolução do modelo gerado pela ferramenta *Web Guitar*, visto que a completude desse modelo influencia diretamente na qualidade dos testes gerados. Nesse contexto, o objetivo foi melhorar o modelo de forma a gerar mais casos de teste.

Como contribuição, a *WG-Modificada* pode ser utilizada por equipes de teste de software para automatizar as atividades de geração e execução de casos de teste. Neste trabalho, a *Web Guitar* foi utilizada para teste de Sistema, porém a mesma pode ser utilizada para outros níveis de teste, como Unidade e Integração. Esses níveis, por possuírem um escopo reduzido, podem facilitar o gerenciamento dos casos de testes gerados devido à grande quantidade de casos de teste gerados pela ferramenta. Além disso, a *WG-Modificada* também oferece suporte a teste de Regressão para aplicações Web, visto que ela gera arquivos de estados que posteriormente podem ser comparados com arquivos resultantes de outras versões da aplicação testada.

5.2. Contribuições

As principais contribuições oferecidas por esta pesquisa à comunidade de teste de software são:

- Evolução e disponibilização da ferramenta *WG-Modificada* que poderá ser utilizado com base para futuras pesquisas na área.
- Base de conhecimento sobre as características relevantes de Teste Baseado em Modelo no contexto de Aplicações Web, que poderá ajudar pesquisadores que desejam trabalhar com essas áreas.

5.3. Limitações

Algumas limitações foram observadas no decorrer da pesquisa:

- Problema na identificação de componentes gerados dinamicamente ou que não possuam identificação (ID e nome da tag);
- Problemas com algumas funções *JavaScript* em que a ferramenta identifica essas funções, mas não permite acessar a página a partir da URL gerada (URL+ função *JavaScript*);
- Problema de preenchimento de dados de entrada que torna a execução da ferramenta mais demorada conforme aumenta-se a quantidade de componentes de entrada da página;
- A forma atual da configuração dos parâmetros iniciais para execução do *Web Ripper* é feita manualmente e, com isso, gasta-se muito tempo executando-se essa tarefa, principalmente se o testador não possui conhecimento da aplicação testada.
- Não execução dos casos de teste gerados por ambas as ferramentas utilizando *Web Replayer*, em virtude do tempo elevado e necessidade de recursos computacionais requeridos por ambas ferramentas, porém é um passo a ser concluído nesta pesquisa.

5.4. Trabalhos futuros

Como trabalhos futuros tem-se:

- Integração com o trabalho de SCHRAMM (2014) do grupo Experts para facilitar a geração dos dados de teste e reduzir o esforço de sua utilização;
- Aprimorar o conversor de grafo EFG para reduzir o problema de identificação de diferentes instâncias de uma mesma página;
- Reduzir o tempo de preenchimento de dados de entrada;
- Utilizar outros algoritmos para percorrer as páginas da aplicação como busca em largura ou *backtraking*.
- Completar o processo de testes da ferramenta *Web Guitar*, seguindo com a execução do *Web Replayer* para os casos de teste gerados por ambas as ferramentas, e analisar a taxa de detecção de defeitos dos novos casos de teste gerados pela *WG-Modificada*.

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, P; MENZ, N.; RATY, T.; SCHIEFERDECKER, I. (2011) “Automated Java GUI Modeling for Model-Based Testing Purposes**”.** 8th International Conference on Information Technology: New Generations (ITNG 2011).

AHO, P.; SUAREZ, M.; KANSTREN, T.; MEMON, A.M. (2013) "Industrial adoption of automatically extracted GUI models for testing". Workshop on Experiences and Empirical Studies in Software Modelling (EESSMod), pp. 49-54.

AHO, P.; SUAREZ, M.; KANSTREN, T.; MEMON, A.M. (2014) "Murphy Tools: Utilizing Extracted GUI Models for Industrial Software Testing". IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp.343-348.

AMALFITANO, D.; FASOLINO, A. R.; CARMINE, S. D; MEMON, A.; TRAMONTANA, P. (2012) “Using GUI Ripping for Automated Testing of Android Applications”. 27th IEEE international conference on Automated software engineering. Washington, IEEE Computer Society.

AMALFITANO, D.; FASOLINO, A.R.; POLCARO, A.; TRAMONTANA, P. (2014)". The DynaRIA tool for the comprehension of Ajax web applications by dynamic analysis". Innovations in Systems and Software Engineering, v.10, Issue 1, pp 41-57.

ANISETTI, M.; ARDAGNA, C.A; DAMIANI, E; SAONARA, F. (2013) *“*A test-based security certification scheme for web services”. ACM Transactions on the Web (TWEB), v.7 n.2, p.1-4.

ARORA, A.A; SINHA, M.B. (2013) “A sustainable approach to automate user session based Máquina de Estados generation for AJAX web applications”. Journal of Theoretical and Applied Information Technology.

ATHIRA, B.A; SAMUEL, P.B. (2010) “Web services regression test case prioritization”. International Conference on Computer Information Systems and Industrial Management Applications (CISIM).

BASILI, V.R.; CALDIERA, G.; ROMBACH, H.D. (1994) “Goal Question Metric Approach”. Ⅱ Encyclopedia of Software Engineering, John Wiley & Sons, Inc, pp. 528-532.

BANSAL, P.; SABHARWAL, S. (2013) “A model based approach to test case generation for testing the navigation behavior of dynamic web applications**”.** Sixth International Conference on Contemporary Computing (IC3).

BELLI, F.; LINSCHULTE, M. (2010) “Event-driven modeling and testing of real-time web services”. Service Oriented Computing and Applications. Volume 4, Issue 1, pp 3-15.

BELLI, F.; BEYAZIT, M; GÜLER, N. (2011) “Event-Based GUI Testing and Reliability Assessment Techniques-An Experimental Insight and Preliminary Results”. Proceedings of IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, ser. ICSTW 11.

BELLI F.; ENDO, A.T; LINSCHULTE, M.; SIMÃO, A. (2012) “A holistic approach to model-based testing of web service compositions”. Software: Practice and Experience.

BERTOLINI, C.; MOTA, A. (2010) “A Framework for GUI Testing Based on Use Case Design”. Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, p.252-259.

BOLIS, F.; GARGANTINI, A.; GUARNIERI, M.; MAGRI, E.; MUSTO, L. (2012.) “Model-driven testing for web applications using abstract State machines”. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

BOUMIZA, D.S.A; AZZOUZ, A.B.B. (2012) “Design and development of a user interface to customize web testing scenarios”. International Conference on Education and e-Learning Innovations (ICEELI).

BOZIC, J.; WOTAWA, F. (2013) “XSS pattern for attack modeling in testing”. 8th International Workshop on Automation of Software Test (AST).

BRYCE, R.C.A; SAMPATH, S.B; MEMON, A.M.C. (2011) “Developing a single model and test prioritization strategies for event-driven software”. IEEE Transactions on Software Engineering.

CARTAXO, E.G. (2006) “Geração de Casos de Teste Funcional para Aplicações de Celulares”. Dissertação de Mestrado em Ciência da Computação. Universidade Federal de Campina Grande. Paraíba.

CASADO, R.; TUYA, J.; YOUNAS, M. (2012) “Testing the reliability of web services transactions in cooperative applications”. Proceedings of the ACM Symposium on Applied Computing.

CHERNONOZHKIN, S. K. (2001) “Automated test generation and static analysis”. Programming and Computer Software, v. 27, n. 2, p. 86–94.

COPSTEIN, B.; OLIVEIRA, F. (2005) “Automated Test Script Generation for Model-Based Testing”. Brazilian Symposium on Software Quality.

DA COSTA, A.D.; VENIERIS, R.; DE CARVALHO, G.R.; DE LUCENA, C.J.P. (2013) “RSA-MBT: A test tool for generating test artifacts based on models”. 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2013.

DALAL, S.R.; JAIN, A.; KARUNANITHI, N.; LEATON, J.M.; LOTT, C.M.; PATTON, G.C.; HOROWITZ, B.M. (1999) “Model-Based Testing in Practice”. Proc. Int. Conf. Software Eng. (ICSE), pp. 285-294.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (2007) “Introdução ao Teste de Software”, Rio de Janeiro: Elsevier.

DI LUCCA, G.A; FASOLINO, A.R. (2006) "Testing Web-based applications: The state of the art and future trends". Information and Software Technology, vol. 48, no. 12, pp. 1172–1186. 2006.

DIAS-NETO, A.C. (2009) “Seleção de Técnicas de Teste Baseado em Modelos”. Programa de Engenharia de Sistemas e Computação. Tese de Doutorado, COPPE. Universidade Federal do Rio de Janeiro.

DIAS-NETO, A. C.; TRAVASSOS, G.H. (2010) “A Picture from the Model-Based Testing Area: Concepts, Techniques, and Challenges”. Advances in Computers, v. 80, p. 45-120.

EDWARDS, S.H. (2001) “A Framework for Practical, automated black-box testing of component-based Software”. Software Testing, Verification and Reliability, Vol. 11, p. 97-111.

ESCALONA, M.J.; TORRES, J.; MEJÍAS, M.; GUTIÉRREZ, J.J.; VILLADIEGO, D. (2007) “The Treatment of Navigation in Web Engineering”. Advances in Eng. Software, vol. 38, pp. 267-282.

ESCOTT, E; STROOPER, P.; STEEL, J.; KING, P. (2011) “Integrating Model-Based Testing in Model-Driven Web Engineering”. Proceedings of the Eighteenth Asia-Pacific Software Engineering Conference.

FANTINATO, M.; DA CUNHA, A.C.R.; DIAS, S.V., CARDOSO, S.A.M.; CUNHA, C.A.Q. (2004) “AutoTest. Um Framework Reutilizável para a Automação de Teste Funcional de Software”. III Brazilian Symposium on Software Quality, p. 286-300.

FASOLINO, A.R.; AMALFITANO, D.; TRAMONTANA, P. (2013) “Web Application Testing in Fifteen Years of WSE”. 15th IEEE International Symposium on Web Systems Evolution (WSE).

FRAJTAK, K.; BURES, M; JELÍNEK, I. (2011) Manual testing of web software systems supported by direct guidance of the tester based on design model. World Academy of Science, Engineering and Technology.

FRAJTAK K.; BURES, M; JELÍNEK, I. (2012) “Formal specification to support advanced model based testing”. Federated Conference on Computer Science and Information Systems, FedCSIS.

FRANCISCO, M.A; LÓPEZ, F.M; FERRERO, H.; CASTRO, L.M. (2013) “Turning web services descriptions into QuickCheck models for automatic testing”. Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP.

FRASER, G.; ARCURI, A. (2012) “Sound empirical evidence in software testing”. Proceedings of the 34th International Conference on Software Engineering.

GANOV, S.; KILLMAR, C.; KHURSHID, S.; PERRY, D.E. (2009) "Event Listener Analysis and Symbolic Execution for Testing GUI Applications". Proceedings of the 11th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering.

GAROUSI, V.; MESBAH, A.; BETIN-CAN, A.; MIRSHOKRAIE, S. (2013) “A systematic mapping study of web application testing”. Information and Software Technology, vol. 55, no. 8, pp. 1396–1374.

GEORGE, C.; KRISHNAN, P.; SALAS, P.A.P.; SANDERS, J.W. (2007) “Specification for testing, Lecture Notes in Computer Science” (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

GUITAR: Disponível em: http://guitar.sourceforge.net/, acessado em: 12/02/2014.

HAJIABADI, H.; KAHANI, M. (2011) “An automated model based approach to test web application using ontology”. IEEE Conference on Open Systems, ICOS.

HAUPTMANN, B.; JUNKER, M. (2011) “Utilizing user interface models for automated instantiation and execution of system tests”. International Workshop on End-to-End Test Script Engineering, ETSE.

HOSSEN, K.; GROZ, R.; ORIAT, C.; RICHIER, J.L (2013) “Automatic generation of test drivers for model inference of web applications”. IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, ICSTW.

HOU, Ying; CHEN, Rang; DU, Zhenjun. (2009) “Automated GUI Testing for J2ME Software Based on FSM”**.** International Conference on Scalable Computing and Communications.

ISABEL, S.L.S. (2011) “Seleção de abordagens de teste para aplicações web”. Programa de Engenharia de Sistemas e Computação. Dissertação de Mestrado, COPPE. Universidade Federal do Rio de Janeiro.

JURISTO, N.; MORENO, A.M.; STRIGEL, W. (2006) "Guest Editors' Introduction: Software Testing Practices in Industry". vol. 23, no. 4, pp. 19-21.

KANER, C.; FALK, J.; NGUYEN, H.Q. (1999) “Testing Computer Software”, 2 ed.

KANER, C. (2006) “Exploratory Testing”. Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando.

KAPPEL, G. (2004) “Web Engineering - Old Wine in New Bottles? Invited Talk”. Fourth International Conference on Web Engineering (ICWE’04).

KEUM, C.A; KANG, S.B. (2011) “Utilizing user interface models for automated instantiation and execution of system tests”. International Workshop on End-to-End Test Script Engineering, ETSE.

KITCHENHAM, B.; CHARTERS, S. (2007) “Guidelines for Performing Systematic Literature Reviews in Software Engineering”. version 2.3. Technical Report, Evidence-Based Software Engineering (EBSE).

KOCH, N.; KNAPP, A.; ZHANG, G.; BAUMEISTER, H. (2008) “Uml-Based Web Engineering - An Approach Based on Standards”. Web Engineering 2008: 157-191.

KOOPMAN, P; ACHTEN, P.; PLASMEIJER, R. (2006) “Model-based testing of thin-client web applications”. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

KOOPMAN, P; ACHTEN, P.; PLASMEIJER, R. (2007) “Model-based testing of thin-client web applications and navigation input”. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

KUEMPER, D.; REETZ, E.; TONJES, R. (2013) “Test derivation for semantically described IoT services”. Future Network and Mobile Summit”. Future Network Summit.

LEMOS, O.A.L.; ZANICHELLI, *F.C.;* RIGATTO, *R.;* FERRARI, *F;* GHOSH, *S.* (2013) “Visualization, Analysis, and Testing of Java and AspectJ Programs with Multi-Level System Graph”. Brazilian Symposium on Software Engineering.

LENZ, C; CHIMIAK–OPOKA, J.; BREU, R. (2007) “Model driven testing of SOA-based software”. Proceedings of CEUR Workshop.

LI, N.; MA, Q.; WU, J.; JIN, M.; LIU, C. (2006) “A Framework of Model-Driven Web Application Testing”. Proceedings of the 30th Annual International Computer Software and Applications Conference, p.157-162.

LI, N.; MA, Q.; WU, J.; JIN, M.; LIU, C. (2007) “Web application model recovery for user input validation testing”. 2nd International Conference on Software Engineering Advances, ICSEA.

LEWIS, E.W. (2000) “Software Testing and Continuous Quality Improvement”. CRC Press LLC.

MAFRA, S.N., BARCELOS, R.F., TRAVASSOS, G.H. (2006) “Aplicando uma Metodologia Baseada em Evidência na Definição de Novas Tecnologias de Software”. Proceedings of the 20th Brazilian Symposium on Software Engineering (SBES 2006), v. 1, pp. 239-254.

MARIANI, L.; PEZZÈ, M; RIGANELLI, O.; SANTORO, M. (2012) “AutoBlackTest: Automatic black-box testing of interactive applications”. Int'l. Conf. Softw. Test., pp. 81-90.

MARIN, B.; VOS, T.; GIACHETTI, G.; BAARS, A.; TONELLA, P. (2011) "Towards testing future Web applications". Fifth International Conference on Research Challenges in Information Science (RCIS), vol., no., pp.1,12, 19-21.

MEMON, A. M., POLLACK, M.E.; SOFFA, M.L. (1999) “Using a goal-driven approach to generate test cases for GUIs”. Proceedings of the 21st International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 257–266.

MEMON, A.; POLLACK, M.E.; SOFFA, M.L. (2001) “Hierarchical GUI test case generation using automated planning”. IEEE Transactions on Software Engineering 2001; 27(2): p.144-155.

MEMON, A.; POLLACK, M.E.; SOFFA, M.L. (2003) "Regression testing of GUIs". Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering.

MESBAH, A.; DEURSEN, A.V.; LENSELINK, S. (2012).” Crawling Ajax-based Web Applications through Dynamic Analysis of User Interface State Changes”. ACM Transactions on the Web *(TWEB).*

MIAO, Y.; YANG, X. (2010) "An FSM based GUI test automation model". 11th International Conference on Control Automation Robotics & Vision (ICARCV), pp.120,126.

MOREIRA, R.M.L.; PAIVA, A.C.R. (2014) "PBGT Tool: An Integrated Modeling and Testing Environment for Pattern-Based GUI Testing". 29th IEEE/ACM International Conference on Automated Software Engineering (ASE).

MYERS, G.J. (2004) “The Art of Software Testing”. 2 ed., John Wiley & Sons, Nova Jersey.

NGUYEN, B.; ROBBINS, B.; BANERJEE, I.; MEMON, A. (2013) Guitar: an innovative tool for automated testing of gui-driven software, Automated Software Engineering, pp. 1-41.

OLIVEIRA, R.B. (2007) “Framework Functest: Aplicando padrões de software na automação de testes funcionais”. Dissertação de Mestrado. Universidade de Fortaleza.

PEREZ, I.R.D.C.*;* MARTIN, E. (2007) “Automação em Projeto de Testes Usando Modelos UML”. Brazilian Workshop on Systematic and Automated Software Testing.

PINHEIRO, P.V.P.; ENDO, A.T.; SIMÃO, A. (2013) “Model-Based Testing of RESTful Web Services Using UML Protocol State Machines”. Brazilian Workshop on Systematic and Automated Software Testing.

PINTO, T.D.; STAA, A.V. (2013) “Functional Validation Driven by Automated Tests”. Brazilian Symposium on Software Engineering.

PRESSMAN, R. (2010) “Engenharia de Software”. McGraw Hill. 8 ed. São Paulo: Porto Alegre: Pearson Mcgraw-Hill: Bookman.

QIAN, L.L.A.B; CHEN, Y.H.A.B. (2011) “Generating test case specifications of web service composition using model checking”. Journal of Shanghai University.

RAUF, A.; ANWAR, S.; JAFFER, M.A.; SHAHID, A.A. (2010) “Automated GUI Test Coverage Analysis using GA”. Seventh International Conference on Information Technology (ITNG 2010).

REZA, H.; OGAARD, K.; MALGE, A. (2008) “A model Based Testing Technique to Test Web Applications Using Statecharts”. Proceedings of the Fifth International Conference on Information Technology: New Generations, ser. ITNG'08. IEEE Computer Society, pp. 183-188.

SANTOS-NETO, P.; RESENDE, R. and PÂDUA, C. (2007) “Requirements for information systems model-based testing”. Proceedings of the 2007 ACM symposium on Applied computing, pp. 1409-1415.

SCHRAMM, B. (2014) “Geração Automática de casos de teste a partir da Web”. Monografia do curso de Sistemas de Informação. Universidade Federal do Amazonas.

SCHUR, M.; ROTH, A.; ZELLER, A. (2013) “Mining behavior models from enterprise web applications”. 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE.

SHAHZAD, A.; RAZA, S.; AZAM, M.N.; BILAL, K.; INAM-UL-HAQ; SHAMAIL, S. (2009) “Automated optimum test case generation using web navigation graphs”. International Conference on Emerging Technologies, ICET.

SHULL, F.; CARVER, J.; TRAVASSOS, G. (2001) “An Empirical Methodology for Introducing Software Processes”. Proceedings of the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9), pp. 288-296.

SILVA, C.; LEITE, J.; COELHO, R. (2011) “Testes de Aceitação: Uma Abordagem Baseada na Especificação de Casos de Uso e no Design da Interação”. Brazilian Workshop on Systematic and Automated Software Testing.

SOUZA, T.; CORREA, A.; SCHMITZ, E.; ALENCAR, A. (2012) “Testes Funcionais de Web Services RESTful a partir de Modelos UML”. Brazilian Symposium on Software Quality.

SPINOLA, R.O.; DIAS-NETO, A.C.; TRAVASSOS, G. H. (2008) “Abordagem para Desenvolver Tecnologia de Software com Apoio de Estudos Secundários e Primários”. Experimental Software Engineering Latin American Workshop (ESELAW).

SPRENKLE, S.; POLLOCK, L.; SIMKO, L. (2011) “A study of Usage-Based Navigation Models and Generated Abstract Test Cases for Web Application”. Fourth International Conference on Software Testing, Verification and Validation.

SPRENKLE, S.; POLLOCK, L.; SIMKO, L. (2013) “Configuring effective navigation models and abstract test cases for web applications by analysing user behavior”. Software Testing Verification and Reliability.

TONELLA, P.; MARCHETTO, A.; NGUYEN, C.; JIA, Y.; LAKHOTIA, K.; HARMAN, M. (2012) “Finding the optimal balance between over and under approximation of models inferred from execution logs”. IEEE 5th International Conference on Software Testing, Verification and Validation, ICST.

TONELLA, P.; MARCHETTO, A.; NGUYEN, C.; LAKHOTIA, K.; HARMAN, M. (2013) “Automated generation of state abstraction functions using data invariant inference”. 8th International Workshop on Automation of Software Test, AST.

TÖRSEL, A. M. (2011) “Automated test case generation for web applications from a domain specific model”. Computer Software and Applications Conference Workshops, vol. 0, pp. 137-142.

TÖRSEL, A. M. (2013) “A testing tool for web applications using a domain-specific modelling language and the NuSMV model checker”. IEEE 6th International Conference on Software Testing, Verification and Validation, ICST.

WANG, X.; GUO, L.; MIAO, H. (2008) “An Approach to transforming UML model to FSM model for automatic testing”. International Conference on Computer Science and Software Engineering, CSSE.

XIE, Q.; MEMON, A. (2008) “Using a pilot study to derive a GUI model for automated testing”. ACM Transactions on Software Engineering and Methodology (TOSEM), v.18 n.2, p.1-35.

XIAOCHUN, Z. (2008) “A Test Automation Solution on GUI Functional Test”. 6th IEEE International Conference on Industrial Informatics, INDIN.

XIONG, P.; STEPIEN, B.; PEYTON, L. (2009) “Model-based penetration test framework for web applications using TTCN-3”. Lecture Notes in Business Information Processing.

XU, D. (2011) “A tool for automated test code generation from high-level petri nets”. Proceedings of the 32nd international conference on Applications and theory of Petri Nets.

XU, D.; TU, M.; SANFORD, M.; THOMAS, L.; WOODRASKA, D.; XU, W. (2012) “Automated security test generation with formal threat models”. IEEE Transactions on Dependable and Secure Computing.

XU, D.; XU, W.; BAVIKATI, B.; WONG, W. (2012) “Mining executable specifications of web applications from Selenium IDE tests”. Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability, SERE.

ZAKONOV, A.; SHALYTO, A. (2012) “Extracting EFSMs of web applications for formal requirements specification”. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

ZHANG, X.; TANNO, H.; HOSHINO, T. (2011) “Introducing Test Case Derivation Techniques into Traditional Software Development: Obstacles and Potentialities”. 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW.

ZHENG, Y.; ZHOU, J.; KRAUSE, P. (2007) “An automatic test case generation framework for web services”, Journal of Software.