UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS CORNÉLIO PROCÓPIO
DIRETORIA DE GRADUAÇÃO E EDUCAÇÃO PROFISSIONAL
DEPARTAMENTO DE COMPUTAÇÃO
ENGENHARIA DE COMPUTAÇÃO

GUILHERME RICKEN MATTIELLO

EVOLUÇÃO DE UMA FERRAMENTA DE TESTE BASEADO EM MODELO PARA A INTEGRAÇÃO COM O FRAMEWORK SELENIUM

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO
2016

GUILHERME RICKEN MATTIELLO

EVOLUÇÃO DE UMA FERRAMENTA DE TESTE BASEADO EM MODELO PARA A INTEGRAÇÃO COM O FRAMEWORK SELENIUM

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina Trabalho de conclusão de curso 1, do curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof. Dr. Andre Takeshi Endo

CORNÉLIO PROCÓPIO

2016

Dedico este trabalho à minha família, principalmente aos meus pais, que sempre apoiaram e incentivaram meus estudos, fornecendo todo e qualquer suporte necessário para que eu pudesse me dedicar exclusivamente à minha formação pessoal e profissional.

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Dr. Andre Takeshi Endo, pela paciência, dedicação e comprometimento com essa orientação e principalmente por compartilhar seus conhecimentos, que foram essenciais para a elaboração deste trabalho.

A minha família, especialmente meus pais Névio e Zélia e meu irmão Eduardo, que sempre me incentivaram e estiveram presentes em momentos difíceis, pois sem o apoio recebido seria muito difícil vencer essa jornada.

Aos meus colegas de trabalho, principalmente ao Guilherme, Juliano, Jean, Denílson, Mario, Wellington, Davi e Felipe, e a todos meus amigos, especialmente ao Henrique e Mateus, que estiveram presentes e me acompanharam diariamente influenciando diretamente na minha formação.

Gostaria de registrar, também, minha gratidão aos professores, coordenadores e demais funcionários da Universidade por todo conhecimento, apoio e pela formação recebida.

Enfim, a todos que, em algum momento, contribuíram para a realização deste trabalho.

RESUMO

MATTIELLO, Guilherme Ricken. **Evolução de uma Ferramenta de Teste Baseado em Modelo para a Integração com o Framework Selenium.** 2016. 29 f. Trabalho de Conclusão de Curso (Graduação) – Engenharia de Software. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

Teste Baseado em Modelo (TBM) é uma abordagem que permite verificar sistemas por meio de casos de testes gerados automaticamente a partir de modelos de teste que foram elaborados com base nos requisitos do software. Neste contexto, é importante que ferramentas de TBM sejam integradas com *frameworks* de teste amplamente utilizados na prática. Este trabalho propõe a evolução da ferramenta MBTS4MA (*Model-Based Test Suite For Mobile Applications*) de TBM, com foco em dispositivos móveis, adaptando-a para testar aplicações *web*. Para isso, será utilizado o framework *Selenium WebDriver* que executa testes implementados em linguagens de programação suportadas (neste projeto, optou-se pelo Java) para os principais *browsers* do mercado, no nível da interface gráfica do usuário da aplicação *web*.

Palavras-chave: Teste baseado em modelo. Teste de software. Grafo de Sequência de Eventos. Selenium. Aplicação *web*. Automatização de teste.

ABSTRACT

MATTIELLO, Guilherme Ricken. **Evolução de uma Ferramenta de Teste Baseado em Modelo para a Integração com o Framework Selenium**. 2016. 29 f. Trabalho de Conclusão de Curso (Graduação) – Engenharia de Software. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

Model-Based Testing (MBT) is an approach that allows checking systems through test cases, automatically generated from test models that were designed based on the software requirements. In this context, it is important that MBT tools are integrated with test frameworks widely used in practice. This work proposes the improvement of the MBT tool MBTS4MA (Model-Based Test Suite For Mobile Applications), that focuses on mobile devices, adapting it to test web applications. For this purpose, the Selenium WebDriver framework will be applied, which performs tests implemented in some supported programming languages (in this case Java was chosen) to the main browsers on the market, on the level of the web application's GUI.

Keywords: Model-Based Testing. Event Sequence Graph. Selenium. Web Application. Test Automation.

SUMÁRIO

1 INTRODUÇÃO

Aplicações *web* estão cada vez mais complexas e mais presentes nas empresas, assumindo funções críticas que se falharem podem causar prejuízos. Isso faz com que a garantia da qualidade e da confiabilidade no desenvolvimento destes produtos seja essencial, e a busca por métodos e técnicas de testes para essas aplicações seja crescente (EMER et al., 2007). Em vista dessa necessidade de assegurar a qualidade dos produtos de software, é importante analisar as técnicas de teste e identificar métodos que atendam essas demandas para que o resultado seja satisfatório.

Segundo Orso e Rothermel (2014), teste de software é uma das abordagens mais praticadas para avaliar a qualidade do software e continua a ser um dos temas mais pesquisados na engenharia de software. Isso se deve pelo fato de que a atividade de teste é, frequentemente, responsável por mais de 50% do custo total de desenvolvimento de um software, portanto são claros os benefícios em aprimorar as técnicas, automatizando-as, e reduzir os custos envolvidos no teste de software (ANAND et al., 2013).

O teste baseado em modelo (TBM) é uma das abordagens de teste de software, e consiste na automatização de testes apoiados na documentação de requisitos de um sistema, criando, para isso, um modelo do comportamento esperado pelo *system under test* - SUT (UTTING et al., 2006).

Essa modelagem do comportamento de um sistema consiste em uma análise das relações entre os eventos, representando-os através de grafos, de modo que possa se estabelecer uma série de testes que irão abranger o grafo (modelos de teste), exercitando todas as relações definidas entre os eventos, identificando, então, as falhas do sistema (PRESSMAN, 2006).

Uma das técnicas de elaboração de modelos de teste é o *Event Sequence Graphs* (ESGs), que representa as interações do usuário e os comportamentos observados com o SUT, por meio de eventos. O grafo deste modelo do SUT é a base do processo de aplicação do TBM (BELLI et al., 2006b).

Muitas ferramentas dão suporte à abordagem de TBM, auxiliando desde a criação dos modelos e casos de teste até a concretização dos casos de teste e execução dos testes. Entre as ferramentas, o MBTS4MA (FARTO, 2016) é uma ferramenta que auxilia em todas as fases da abordagem TBM para testes em

aplicações para a plataforma Android. Nessa ferramenta, o usuário insere como entrada o projeto do SUT Android, do qual são extraídos metadados que auxiliam na criação dos modelos de teste. A saída é um projeto de teste para o SUT.

1.1 MOTIVAÇÃO

O desenvolvimento de software não é uma tarefa simples e, dependendo do tamanho e da complexidade do software, pode se tornar uma tarefa muito complexa, ficando sujeita a diversos problemas que levam a resultados diferentes do esperado, muitos destes problemas causados por erros humanos (DELAMARO et al., 2007).

Em adicional, nos últimos anos, os softwares produzidos têm ficado cada vez maiores e mais complexos, com componentes desenvolvidos por diferentes colaboradores, em diferentes versões, técnicas e linguagens de programação, e ainda para diversas plataformas (ANAND et al., 2013). Essa diversidade de componentes pode ser observada em softwares desenvolvidos para plataforma *web*, que são dinâmicos e heterogêneos. Essas características das aplicações *web* são consequência das várias tecnologias envolvidas em seu desenvolvimento e das diversas maneiras de comunicação entre os componentes que a compõem, o que dificulta a busca pela garantia de qualidade em comparação com outras aplicações tradicionais (EMER et al., 2007).

Neste cenário, o teste de software, um conjunto de atividades que podem ser planejadas e conduzidas sistematicamente com o intuito de identificar defeitos que foram introduzidos ao produto no momento em que foi projetado e construído, ganha espaço na engenharia de software e na busca pela qualidade dos produtos gerados.

O teste é uma etapa do desenvolvimento de software que, frequentemente, responde por mais esforço no projeto que as outras etapas e atividades. Dessa forma, se realizado sem planejamento, tempo e esforço são desperdiçados, e consequentemente erros são agregados ao software despercebidamente. Portanto, é necessário traçar uma estratégia sistemática e utilizar técnicas que assegurem a qualidade do produto desenvolvido (PRESSMAN, 2006).

Uma das abordagens que, segundo Orso e Rothermel (2014), tem sido considerada um sucesso é o TBM, e o grande interesse da comunidade nesse paradigma é comprovado pela vasta quantidade e diversidade de ferramentas,

gratuitas e comerciais, que suportam TBM[1]. Além disso, oferece muitas vantagens, como facilitar o entendimento do funcionamento e da arquitetura de um produto, possibilitando, inclusive, identificar defeitos na elicitação dos requisitos do SUT, agilizar a criação de testes e tornar os testes menos suscetíveis aos erros humanos, por meio de rotinas automáticas (SINHA e SMIDTS, 2006).

Assim, é válida a aplicação da abordagem de TBM para garantir a qualidade de produtos de software, como aplicações *web*, utilizando como apoio um *framework* já consolidado para este tipo de arquitetura, o Selenium WebDriver (SELENIUM, 2016). Esse *framework* executa testes automatizados, escritos em linguagens de programação suportadas, na GUI da aplicação *web*.

1.2 OBJETIVOS

O objetivo deste trabalho é evoluir uma ferramenta de TBM de forma que a mesma fique adequada a aplicações *web*. Em particular, a ferramenta gerada receberá como entrada a URL do SUT e as classes *Page Object*, utilizará a ferramenta MBTS4MA como apoio para gerar os modelos e os casos de teste e o *framework* Selenium WebDriver para a concretização dos casos de teste em códigos executáveis no SUT.

1.3 ORGANIZAÇÃO TEXTUAL

Este trabalho foi dividido da seguinte forma: o Capítulo 2 apresenta o embasamento teórico necessário para ambientação com o tema e está subdividido nos assuntos: aplicações *web*, TBM (elucidando os conceitos de ESG, Selenium WebDriver e apresentando o framework MBTS4MA) e trabalhos relacionados. O Capítulo 3 apresenta a proposta de trabalho, explicando como será conduzido, e o cronograma de execução das atividades propostas.

---

[1] Lista de ferramentas que apoiam TBM: http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html e http://robertvbinder.com/open-source-tools-for-model-based-testing

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado da seguinte forma: na Seção 2.1 será explicado o funcionamento de aplicações *web*, na Seção 2.1.1 será dado um exemplo deste tipo de aplicação, a qual será utilizada para introduzir os outros conceitos envolvidos neste trabalho. Na Seção 2.2 será abordado o conceito de teste, com foco em teste baseado em modelo. Na Seção 2.2.1 será introduzido o *Event Sequence Graph* (ESG) e na Seção 2.2.2 o *framework* Selenium WebDriver, ambos conceitos importantes para o desenvolvimento da proposta deste trabalho. Na Seção 2.3 será apresentada a ferramenta MBTS4MA, utilizada como base para implementar o artefato esperado no resultado deste trabalho. Na Seção 2.4 serão apresentados os trabalhos relacionados.

2.1 APLICAÇÕES WEB

Uma aplicação web é um sistema de software com arquitetura cliente/servidor, que trabalha com requisições e respostas. Em geral, o cliente (normalmente *browsers*) realiza uma requisição, por meio do protocolo HTTP - *Hypertext Transfer Protocol* (W3C, 2016), o servidor web processa o pedido e responde com páginas HTML, CSS, Javascript ou imagens, de forma que o *browser* consiga processar e mostrar ao usuário o conteúdo solicitado (GONÇALVES, 2007).

Uma aplicação *web* pode ser dividida em duas partes principais, o *back-end* e o *front-end*. O *front-end* é a camada responsável pela interface – *Graphic User Interface* (GUI) - que o usuário irá encontrar ao acessar uma aplicação *web*. Nesta camada são utilizadas, principalmente, como tecnologias, as linguagens HTML, CSS e JavaScript. No *back-end* são encontrados códigos que são executados exclusivamente do lado servidor da aplicação, desenvolvidos em linguagens de programação como ASP, PHP e Java (WALES, 2014).

Na Figura 1 está representada a arquitetura cliente/servidor em aplicações *web*.

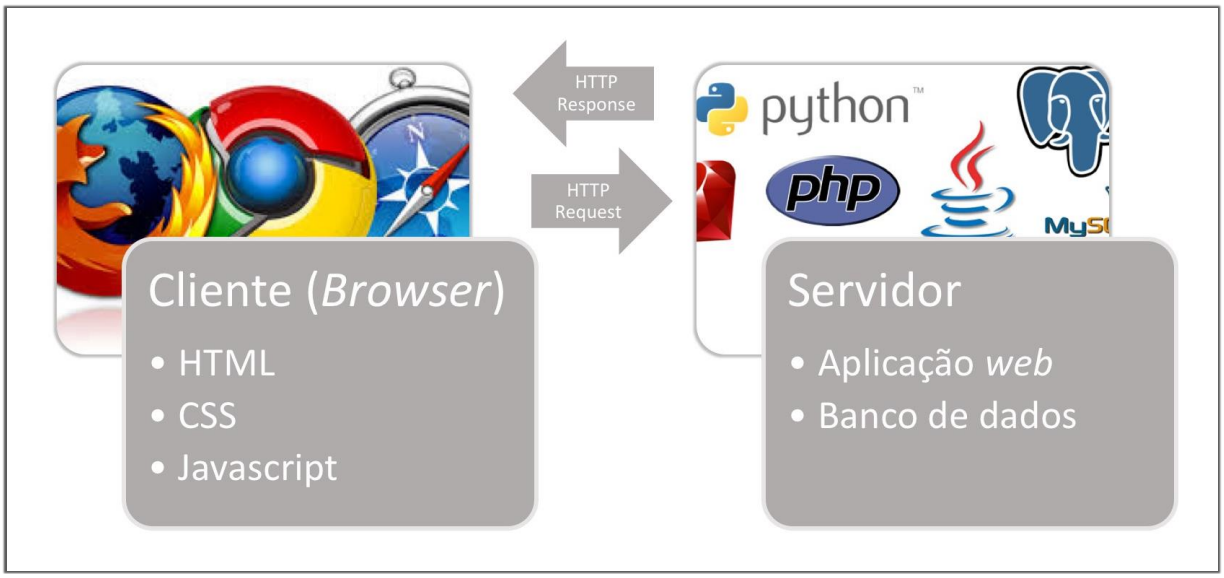

Figura 1: Arquitetura cliente/servidor básica em aplicações *web*
Fonte: Autoria própria

O escopo deste trabalho compreende a realização de testes na GUI (camada *front-end*) das aplicações *web*.

2.1.1 Exemplo

Como exemplo, foi utilizada uma aplicação web que gerencia inscrições em pequenos eventos, como semanas acadêmicas, *workshops* e seminários, na UTFPR, no câmpus Cornélio Procópio. Nesse sistema de eventos, o participante se inscreve, informando dados pessoais (conforme formulário ilustrado na Figura 2), em seguida seleciona os minicursos e palestras que deseja participar (se houver vagas) e finalmente realiza o pagamento. O participante ainda tem as opções de recuperação de senha e de visualizar a programação do evento, através dos menus “Programação” e “Esqueci minha senha”, respectivamente. O sistema possui uma área administrativa, onde os administradores do sistema têm a opção de configurar as informações do evento em questão. Entre essas opções de configuração, está o período de inscrições, cores do *layout* do site, conteúdo da página inicial e programação, e cadastro de minicursos e palestras. O processo de inscrição consiste, seguindo seu fluxo comum, em preencher os seguintes campos com os dados do participante: CPF, nome, *e-mail*, logradouro, número, bairro, estado, cidade, telefone, profissão, usuário, senha. Após preencher todos os campos, clicar no botão “ENVIAR”. Como fluxo alternativo, ao preencher o CPF o sistema verifica se o participante já está cadastrado na base de dados da universidade, e, caso já esteja, o participante não precisa preencher os

campos nome, *e-mail*, usuário e senha, pois já serão recuperados do banco de dados existente.

Esse sistema foi desenvolvido utilizando as linguagens HTML5 (W3C, 2016), JavaScript (ECMA, 2016) e CSS3 (W3C, 2016), e a linguagem de programação PHP (THE PHP GROUP, 2016), com o *framework* Laravel (OTWELL, 2016). O banco de dados utilizado foi o PostgreSQL (THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2016).

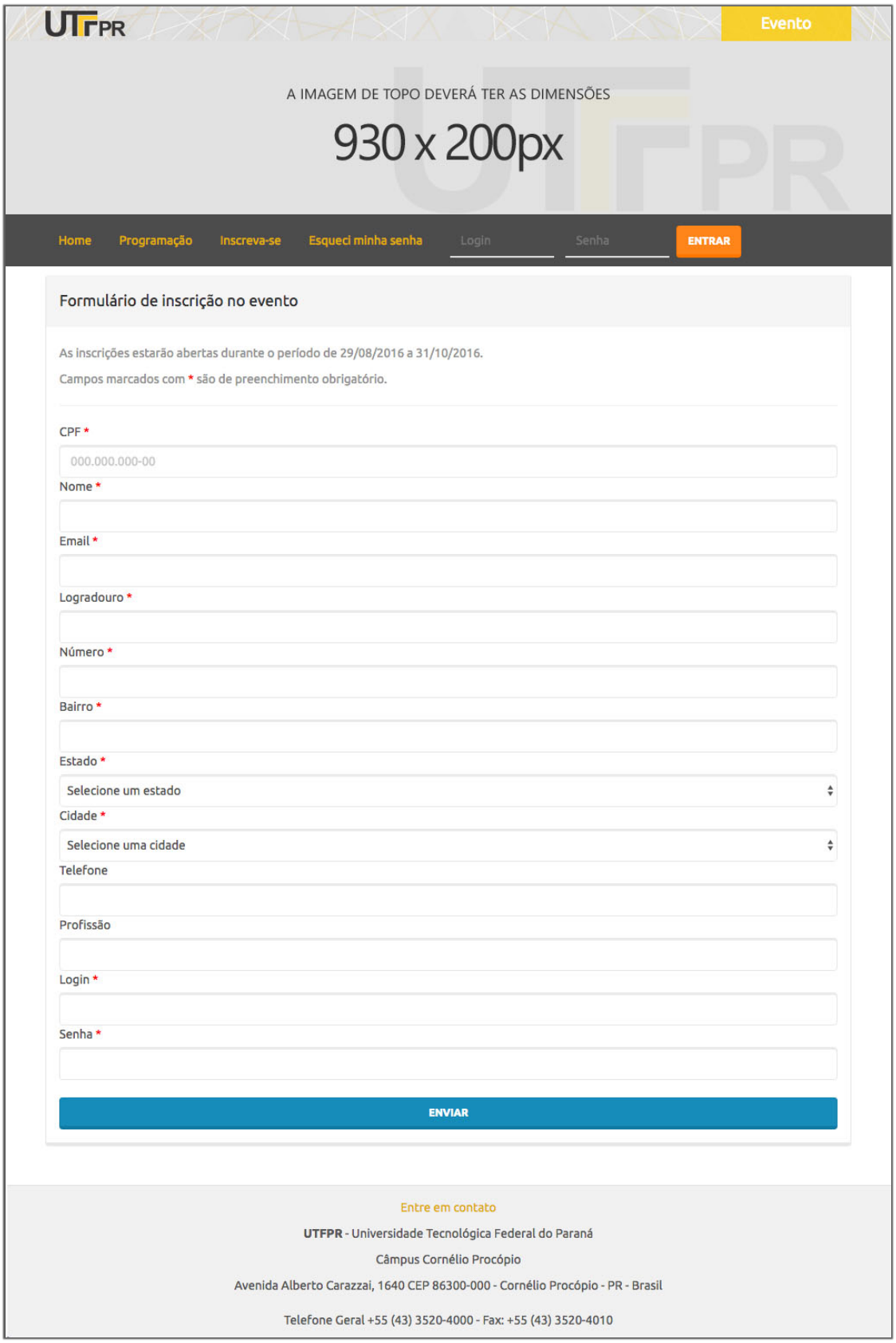

Figura 2: Sistema exemplo - Formulário de inscrição
Fonte: Autoria própria

2.2 TESTE BASEADO EM MODELO

Na atividade de teste de software, mecanismos são implementados para detectar defeitos em software observando e comparando o comportamento real com o esperado, segundo suas especificações. Em um mundo ideal, todas as combinações de entradas e saídas de um sistema deveriam ser testadas, garantindo a consistência desse SUT. Porém, como isso é impraticável, o estudo e a aplicação de técnicas de teste se fazem necessárias para garantir a qualidade de um sistema (MYERS et al., 2012).

Uma destas abordagens é o TBM, que consiste em modelar o comportamento de um sistema, em entradas e saídas, de forma que a saída representada no modelo seja a saída do SUT. Este tipo de teste visa minimizar a prática de testes não estruturados, que dificilmente são reproduzidos, por não serem documentados (UTTING e LEGEARD, 2006). Os modelos de teste de um sistema devem ser verificados de forma a garantir que sejam mais simples que o sistema em teste, senão a dificuldade em validar os modelos gerados seria equivalente a validar o SUT, inviabilizando essa abordagem.

TBM pode ser dividido em quatro etapas principais (ENDO, 2013), representadas na Figura 3, descritas a seguir.

1. Modelagem: Em teste de software é necessário conhecer os requisitos de um sistema e suas funcionalidades, e também características do ambiente em que o sistema está hospedado, como sistema operacional, aplicações e bibliotecas. Nesta etapa, será criado um modelo do SUT, que deve ser menor e mais simples que o SUT, mas que deve conter detalhes que descrevam com acurácia toda a parte do software a ser testada.

2. Geração de casos de teste: Esta etapa depende da técnica de modelagem que descreve os modelos de testes, pois a partir destes modelos serão gerados os casos de teste. Os resultados desta etapa são sequências de operações, extraídas do modelo.

3. Concretização: Os casos de testes gerados na etapa de geração serão transformados de modo a serem executados no SUT, visando automatizar todo o processo. Essa transformação, ou concretização dos casos de teste, é feita utilizando ferramentas de transformação que usam vários *templates* ou mapeamentos para traduzir os casos de teste em códigos executáveis, ou utilizando códigos adaptadores.

Os adaptadores são códigos que basicamente concretizam uma entrada, de modo que fique aplicável ao SUT, aplicam essa entrada ao SUT e abstraem a saída (ao nível do modelo) permitindo a comparação entre o resultado esperado e o resultado de saída do SUT.

4. Execução dos testes: Nesta etapa, os casos de teste, após concretizados, serão executados no sistema que está sendo testado.

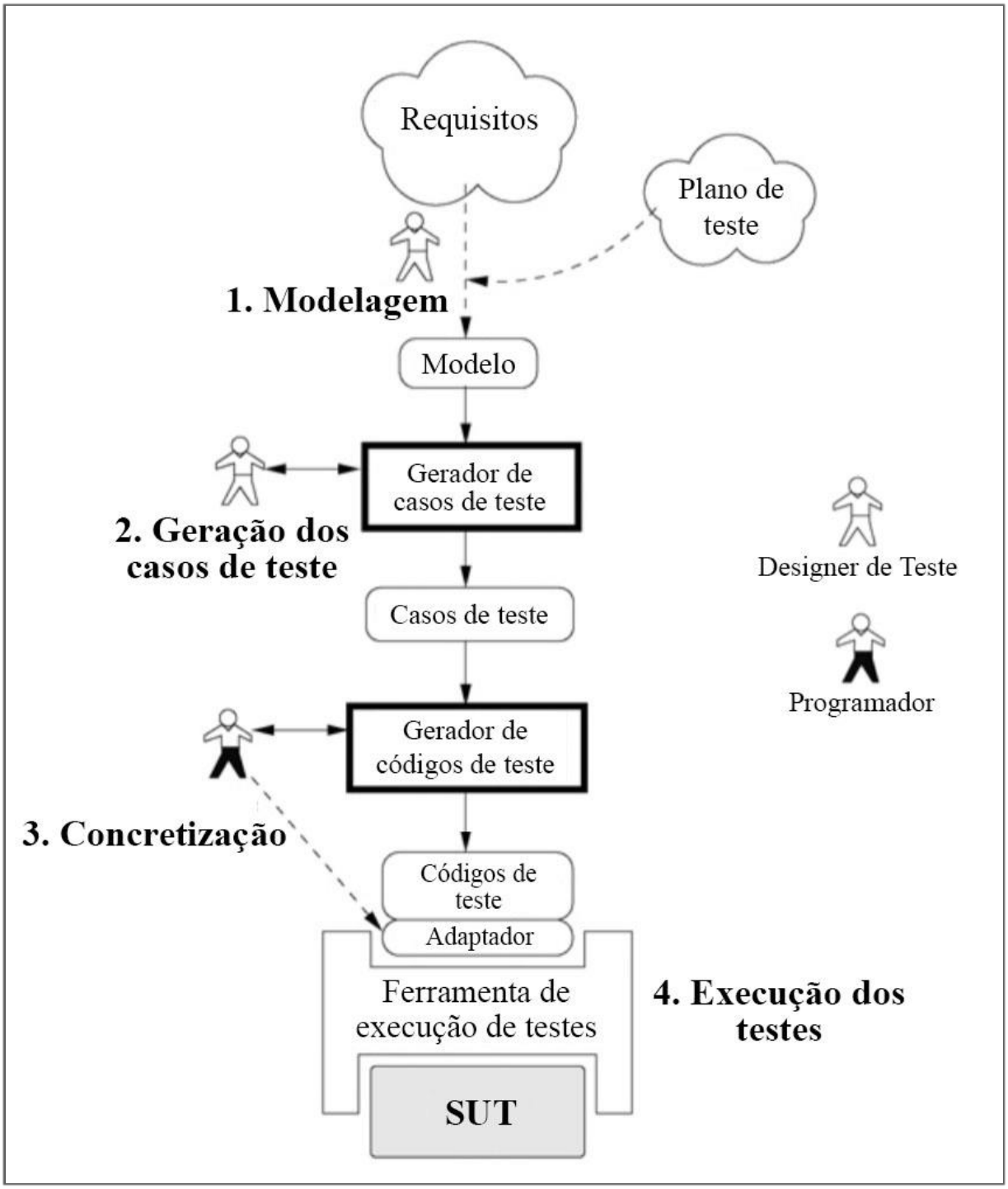

Figura 3: Etapas do TBM
Fonte: Adaptado de UTTING e LEGEARD (2006).

2.2.1 Event Sequence Graph

Segundo Belli et al (2006a), Event Sequence Graph (ESG) é uma técnica de modelagem utilizada para modelar as interações entre um sistema e o usuário que o utiliza, através de eventos. Eventos são fenômenos externamente observáveis, os quais podem ser um estímulo oriundo de um usuário ou uma resposta do sistema, mapeando os diferentes estágios da atividade de um sistema. O ESG é um grafo direcionado, onde os vértices (ou nós) representam as interações usuário/sistema e as arestas (ou arcos) indicam o fluxo possível entre as interações. Para indicar as entradas e saídas de um ESG, os nós de entrada são precedidos de um pseudo vértice com o símbolo “[“ e os nós de saída são seguidos de um pseudo vértice com o símbolo “]”.

Na Figura 4, foi modelado o ESG para o procedimento de inscrição de participantes do sistema de eventos, descrito na Seção 2.1.1.

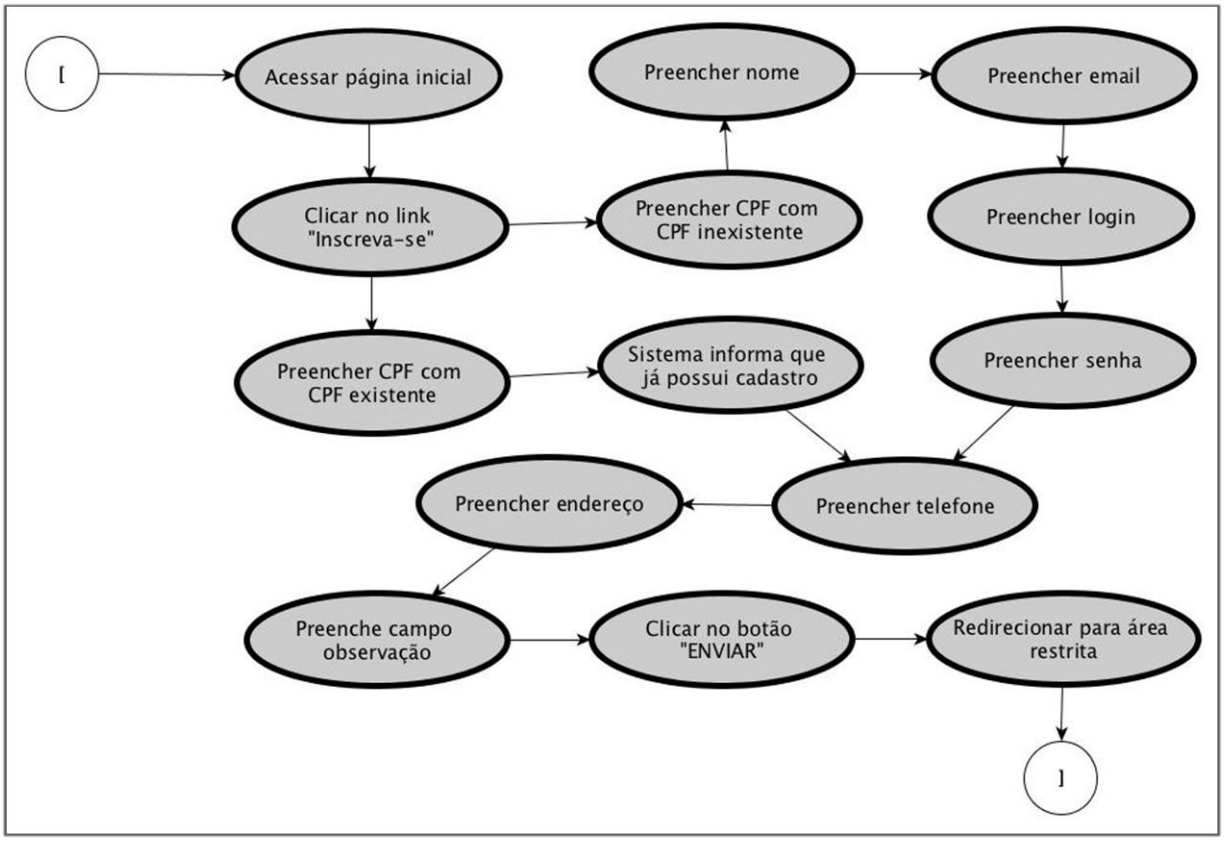

Figura 4: ESG da inscrição no sistema de eventos
Fonte: Autoria própria

Aplicações *web* trabalham com o paradigma de eventos, logo utilizar ESG para modelagem torna-se uma vantagem, já que um ESG permite não só testar o comportamento esperado de um sistema, mas também as interações inesperadas

entre usuário e o SUT. O ESG é muito usado por utilizar notações formais de teorias consolidadas, como teoria dos grafos e teoria dos autômatos. Outra vantagem de um ESG está na característica de ser aprendido em um curto período de tempo e ser suportado por ferramentas específicas a este fim (BELLI et al., 2012).

2.2.2 Ferramenta Selenium (WebDriver)

Selenium (SELENIUM, 2016) é uma ferramenta de automatização de *browsers*, e consequentemente de aplicações *web*, com o objetivo de realizar testes nas aplicações por meio de sua interface gráfica. O Selenium contém a API WebDriver que permite que os testes sejam escritos em linguagens de programação como Java, C#, Ruby, Python, para os *browsers* mais populares, como Google Chrome, Mozilla Firefox e Safari. O Selenium faz chamadas diretas aos *browsers*, através do suporte nativo para automação de cada *browser*, realizada por seu *driver* específico (SELENIUM, 2016).

O *framework* Selenium conta também com o Selenium IDE, um *plugin* para o *browser* Mozilla Firefox, que grava e reproduz as interações entre o usuário e o *browser*. Esta IDE permite ainda a edição e depuração dos testes, sendo muito utilizada para criar códigos para testes no mesmo ambiente em que serão executados.

Atualmente, um dos padrões de projetos utilizados com Selenium é chamado de *Page Object* (FOWLER, 2013), que objetiva separar a especificação da implementação dos testes. Neste padrão, as páginas da aplicação *web*, normalmente, são tratadas como objetos, os quais encapsulam os atributos e métodos referentes a elas. A utilização de *Page Objects* auxilia na evolução da aplicação *web*, pois gera códigos mais robustos e promove o reuso, legibilidade, manutenibilidade e o baixo acoplamento entre o código de teste e a aplicação *web* (LEOTTA et al., 2015).

O código fonte da Figura 5 representa trechos da classe *Page Objects* referente à página de inscrição do sistema de evento. No Selenium, todos os componentes de uma página *web* (*inputs, buttons*, entre outros) precisam ser inicializados antes de serem utilizados. Essa inicialização é feita através da extensão do *Page Objects*, a *Page Factory* (linhas 28 e 29 da Figura 6). A *Page Factory* irá inicializar todas as variáveis do tipo *WebElement* que correspondem a algum elemento da página *web* que foi referenciado por meio dos *locators* (códigos com objetivo de identificar elementos *web*) do Selenium. Neste caso, foi utilizada a anotação

“@FindBy” (linha 25 da Figura 5) que identifica o componente *input* CPF por meio de seu atributo *id*. Ainda na Figura 5, a linha 26 contém a anotação “@CacheLookup” que armazena *locators* frequentemente utilizados (neste caso o *input* CPF) para melhoria de desempenho. Na linha 27 a variável “cpf” do tipo *WebElement* é declarada. As linhas 170 a 173 correspondem ao código que realiza o clique no botão “Enviar” do formulário de inscrição. As linhas 314 a 317 representam o método que preenche o *input* CPF do formulário de inscrição. Os métodos “setCpfTextField()” e “clickEnviarButton()” foram utilizados no código de teste ilustrado na Figura 6.

```
24    ...
25    @FindBy(id = "cpf")
26    @CacheLookup
27    private WebElement cpf;
28    ...
169   ...
170   public Inscricao clickEnviarButton() {
171      enviar.click();
172      return this;
173   }
174   ...
313   ...
314   public Inscricao setCpfTextField(String cpfValue) {
315      cpf.sendKeys(cpfValue);
316      return this;
317   }
318   ...
```

Figura 5: Códigos Page Objects gerados pelo Selenium Page Objects Generator
Fonte: Autoria própria

O código da Figura 6 representa um código de teste que realiza a inscrição de um participante no sistema utilizado como exemplo (sistema de eventos), considerando o ESG da Figura 4. O código foi escrito utilizando os *frameworks* Selenium e JUnit, observando o padrão *Page Objects*, com utilização da classe *Page Object* (nomeada “Inscricao”) onde os trechos de código representados na Figura 5 estão inseridos. A classe “Inscricao” encapsula os métodos e funcionalidades da página *web* que contém o formulário de inscrição.

Na Figura 6, as linhas 19 a 22 contém o método construtor da classe “EventoTest”, o qual inicializa o *driver* (do Selenium WebDriver) do *browser* utilizado

nos testes. O método “setUp()” nas linhas 24 a 30 passa para o *driver* a URL da aplicação *web* e inicializa as páginas “Index” e “Inscricao” que são utilizadas nos testes. Nas linhas 32 a 52 está representado o método principal deste caso de teste, o qual preenche os *inputs* (linhas 36 a 47) e “clica” no botão “Enviar” (linha 49), do formulário de inscrição, finalizando o cadastro. Após o envio do formulário, o código de teste verifica, através de uma assertiva (linha 51), se o cadastro foi realizado com sucesso.

```
12 public class EventoTest {
13
14    public WebDriver driver = null;
15
16    Inscricao inscricao;
17    Index index;
18
19    public EventoTest() {
20       System.setProperty("webdriver.chrome.driver", "/Users/guimat/Desktop/chromedriver");
21       driver = new ChromeDriver();
22    }
23
24    @Before
25    public void setUp() {
26       driver.get("http://eventos.cp.utfpr.edu.br/laraevento/public");
27       driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
28       index = PageFactory.initElements(driver, Index.class);
29       inscricao = PageFactory.initElements(driver, Inscricao.class);
30    }
31
32    @Test
33    public void inscricao(){
34       index.clickInscrevaseLink();
35
36       inscricao.setCpfTextField("04505183963");
37       inscricao.setNomeTextField("Nome do participante");
38       inscricao.setEmail1EmailField("email@dominio.com.br");
39       inscricao.setLogradouroTextField("Rua X");
40       inscricao.setNmeroNumberField("26");
41       inscricao.setBairroTextField("Centro");
42       inscricao.setEstadoDropDownListField("PR");
43       inscricao.setCidadeDropDownListField("Cornélio Procópio");
44       inscricao.setTelefoneTelField("4335204000");
45       inscricao.setProfissoTextField("Estudante");
46       inscricao.setLogin2TextField("usuarioteste");
47       inscricao.setSenha2PasswordField("1234");
48
49       inscricao.clickEnviarButton();
50
51       assertEquals(driver.getCurrentUrl(), "http://eventos.cp.utfpr.edu.br/laraevento/public/");
52    }
53 }
```

Figura 6: Código de teste do procedimento de inscrição utilizando Page Objects
Fonte: Autoria própria

2.3 FERRAMENTA MBTS4MA

Este trabalho propõe aprimorar a ferramenta MBTS4MA (FARTO, 2016) de modo que esta seja utilizável para testar aplicações *web*. Esta ferramenta foi escolhida pelo fato de ser *open source* e por apresentar características semelhantes às desejadas pelo resultado deste trabalho.

A ferramenta MBTS4MA (*Model-Based Test Suite For Mobile Applications*) é uma ferramenta, desenvolvida em Java, que auxilia o processo de automatização de testes para aplicativos móveis, tendo como principal alvo a plataforma Android. Para concretizar os casos de teste gerados o MBTS4MA utiliza a plataforma de teste Robotium, um *framework* para testes automatizados para Android. Dentre as vantagens do uso do *framework*, este permite que sejam elaborados casos de teste de rápida execução, exigem apenas um conhecimento superficial sobre o SUT e os casos de teste gerados são mais robustos pelo fato de se comunicarem com os componentes da interface em tempo de execução.

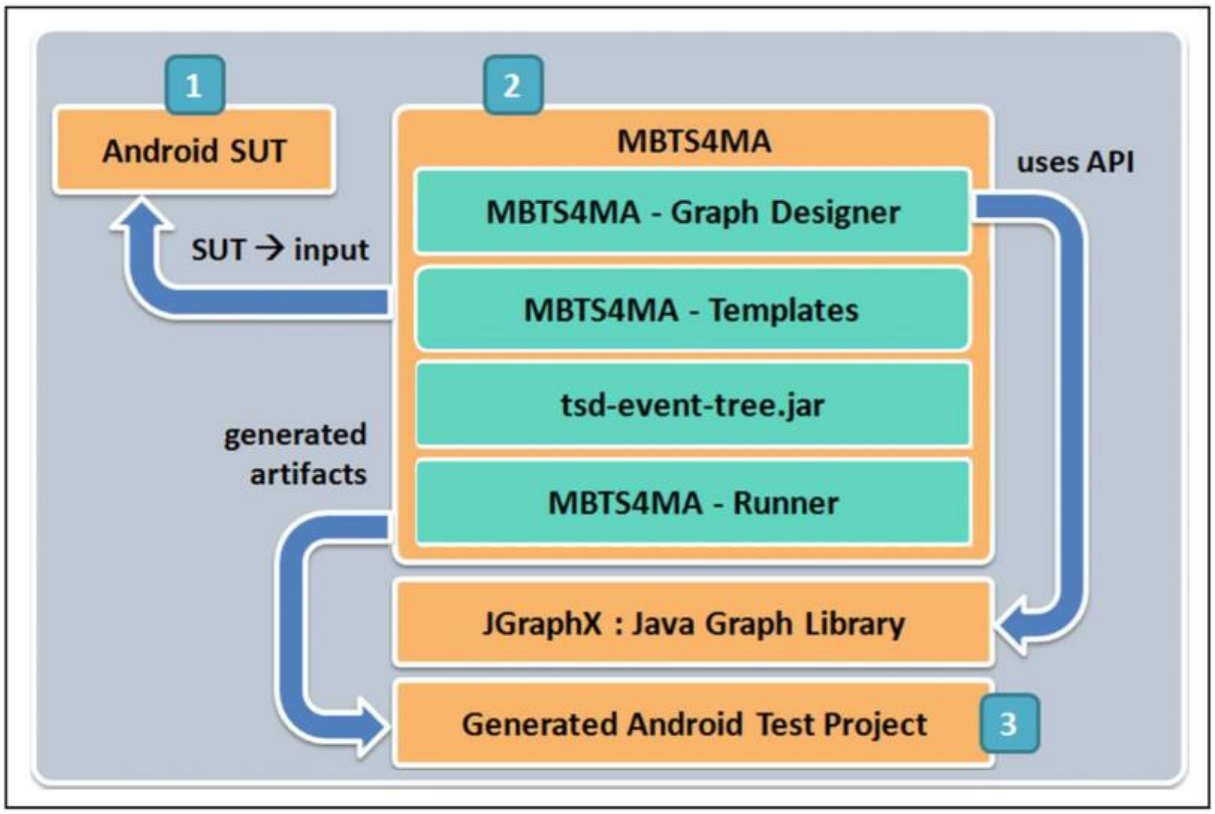

Figura 7: Diagrama arquitetural da ferramenta de apoio MBTS4MA
Fonte: Adaptado de FARTO, 2016

O uso da ferramenta consiste, primeiramente, em informar como entrada a aplicação móvel que será testada, representado na Figura 7 pelo Item 1. Assim, o SUT

passa a ser acessível através da ferramenta, de forma que são extraídos metadados que auxiliarão na elaboração dos modelos de teste.

O Item 2 representa a arquitetura dos componentes da ferramenta MBTS4MA. O *Graph Designer* oportuniza a elaboração de modelos ESG integrados com a aplicação móvel que está sendo testada, já extraindo desta aplicação alguns metadados, como rótulos, nomes de tela e configurações gerais. A camada de *Templates* representa o conjunto de métodos, classes e componentes que fornecem a possibilidade de otimizar e viabilizar a geração de projetos de testes, representando a etapa de concretização dos casos de teste. Ainda no item 2, o tsd-event-tree.jar é uma biblioteca de artefatos e classes, desenvolvida em Java, que implementa uma estrutura de dados baseada em grafos direcionados (ENDO, 2013). A camada *Runner* representa classes e anotações em Java que abstrai o relacionamento entre os casos de testes gerados, e os métodos de teste concretizados que podem ser executados no SUT.

A Figura 8 ilustra o uso da ferramenta MBTS4MA na criação do modelo do SUT e geração automática do fluxo de eventos.

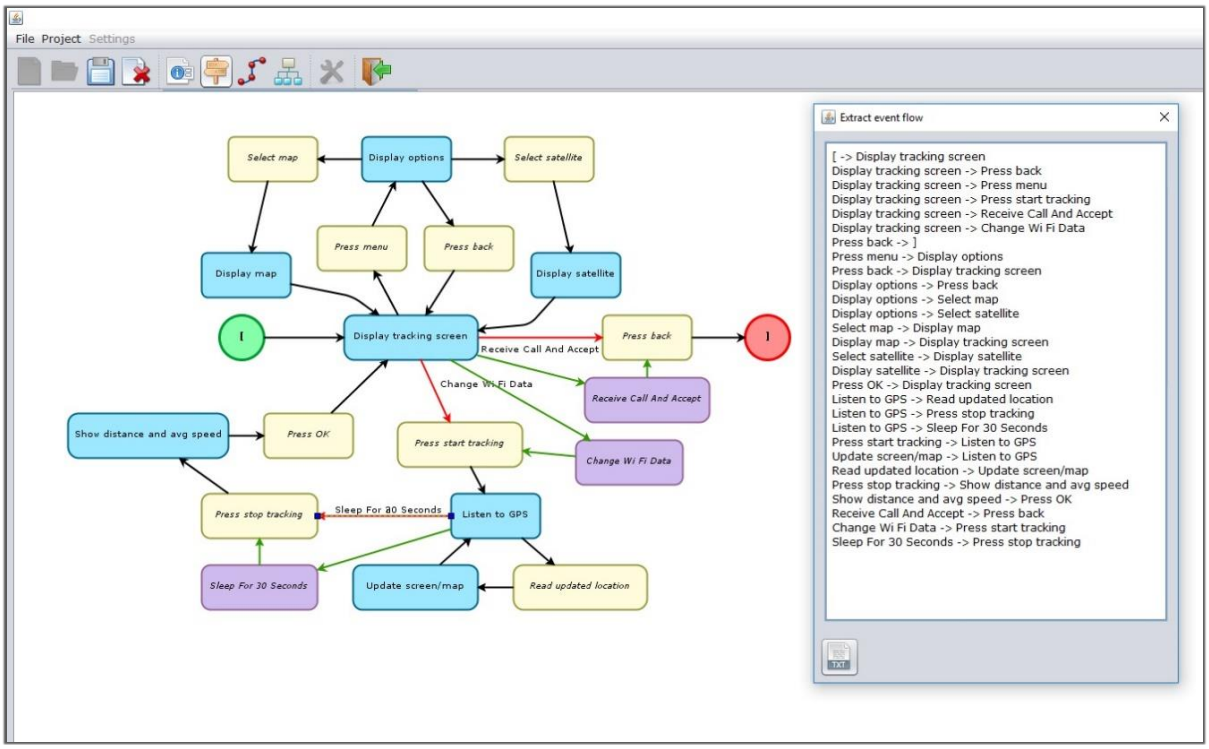

Figura 8: Ferramenta MBTS4MA – Modelagem e geração automática do fluxo de eventos
Fonte: Autoria própria

Em uma visão geral, esse framework suporta a etapa de modelagem e implementa uma relação estreita com o SUT, através da extração de metadados da

aplicação móvel, gerando projetos de teste que podem ser executados em AVDs e/ou dispositivos reais.

O framework MBT4SAMA está disponível, como um projeto *open source*, em https://github.com/guilhermefarto/MBTS4MA

2.4 TRABALHOS RELACIONADOS

Analisando a literatura é possível encontrar trabalhos que estudam o padrão de testes utilizando *Page Objects*. Leotta et al. (2015) implementaram um software que auxilia na criação de códigos de teste, o APOGEN (*Automatic Page Object Generator*), que gera automaticamente *Page Objects* para aplicações *web*. Os autores realizaram uma revisão de literatura e concluíram que no momento não existiam soluções automáticas e efetivas para o problema da geração automática de Page Objects para aplicações *web*.

O APOGEN foi implementado em Java, como uma ferramenta *open source*, e funciona da seguinte maneira: primeiramente, o módulo *Crawler* gera um modelo baseado em estados, com os elementos clicáveis, os estados visitados, as URLs, as *screenshots* da tela e os *links* para outros estados. Essas informações serão processadas pelo módulo *Static Analyser*, o qual irá fazer um mapeamento do SUT para as *PageObjects* que serão criadas, como por exemplo definir nomes únicos para cada classe, gerar os localizadores para cada elemento da página *web*, transformando esses elementos em *WebElement* na classe do *Page Object*. Por fim, o módulo *Code Generator* irá transformar o modelo gerado pelo *Static Analyser* em códigos compreensivos para o *framework* Selenium. A Figura 9 ilustra o esquemático do APOGEN.

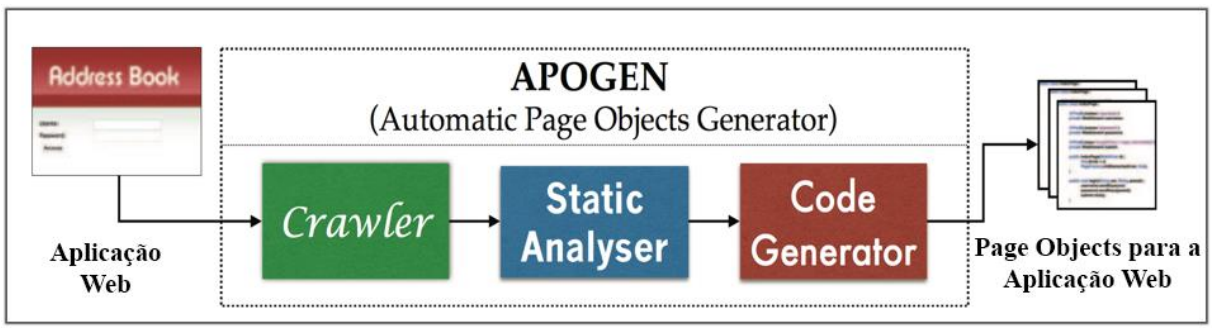

Figura 9: Esquemático da ferramenta APOGEN
Fonte: Adaptado de Leotta et al., 2015

No trabalho, foram comparados um conjunto de *Page Objects* gerados pelo APOGEN, com um conjunto gerado manualmente por um testador humano. Os resultados indicaram que 75% dos códigos gerados pelo APOGEN não necessitaram correções, e os 25% restantes precisaram de alterações mínimas, mas que não inviabilizaram o uso do software.

Outro trabalho encontrado na literatura (CLEAVELAND et al., 2014) faz uma comparação entre teste manual e TBM. Os testes manuais foram dirigidos por um testador de uma empresa de desenvolvimento de software e não contaram com apoio de ferramentas de automatização de casos de teste. Os testes realizados com a abordagem TBM foram dirigidos por um testador de um centro de pesquisa e tiveram a geração e execução de casos de teste de forma automatizada.

Os resultados mostraram que os testes manuais necessitaram de menos tempo de preparação e que os casos de teste não cobriram totalmente o SUT. Por outro lado, o TBM exigiu mais tempo de preparação, foi mais sistemático e detectou mais defeitos. Enquanto os testes realizados manualmente detectaram mais inconsistências na GUI, como rótulos incorretos, o TBM identificou mais problemas funcionais.

2.5 CONSIDERAÇÕES FINAIS

Neste capítulo foram elencados e explicados os conceitos necessários para compreender o problema que este trabalho irá abordar. O crescimento do volume de aplicações *web* no mercado elevou a importância destas nas instituições. Isso tem aumentado a busca por mecanismos e técnicas que garantam a qualidade esperada e que o software cumpra com seu propósito. Assim, o TBM, aliado a ferramentas que automatizam o processo de teste, tem ganhado espaço entre as estratégias adotadas para controle de qualidade dos produtos de software. No Capítulo 3 será apresentada a proposta deste trabalho.

3 PROPOSTA

Com base no conteúdo apresentado no Capítulo 2, o presente trabalho propõe a evolução da ferramenta MBTS4MA, com o intuito de usar TBM em aplicações *web*, apoiado pelo *framework* Selenium WebDriver com códigos de teste escritos na linguagem de programação Java, utilizando o padrão de projetos *Page Objects*.

O artefato gerado por este trabalho consiste em uma ferramenta onde o usuário, que dirigirá os testes ao SUT, inserirá como entrada a URL e as classes *Page Object* do SUT, e a ferramenta extrairá os metadados da aplicação que auxiliarão na elaboração dos testes. Diferentemente do MBTS4MA, onde a entrada é o projeto da aplicação *mobile*, escolheu-se a URL como entrada, pois como os testes são realizados na interface, não é necessário possuir o código fonte do projeto para a extração dos metadados, e isso ainda facilita o uso do *framework* para o usuário. Outro ponto importante é que o usuário poderá inserir como entrada, do *framework*, a URL de um sistema que esteja hospedado em um servidor remoto, levando em consideração a configuração do ambiente de hospedagem do SUT que, segundo Utting e Legeard (2006), é um ponto importante a ser considerado em teste de software.

Junto à URL, o usuário inserirá as classes *Page Objects* para a aplicação, as quais auxiliarão o usuário na criação do modelo de teste através de grafos (ESG), como ilustrado na Figura 9, onde as interações do modelo (ações entre eventos) serão os métodos encapsulados pelas classes *Page Objects*. Neste trabalho, assume-se que as classes *Page Objects* foram geradas em padrão Java, manualmente por algum testador, ou automaticamente por algum *framework*, e serão inseridas como entrada na ferramenta.

Outro *framework* que servirá de apoio é o *JavaParser* (JAVAPARSER, 2016), que permite a manipulação e extração de metadados da estrutura de código fonte Java. Essa ferramenta poderá ser usada para processar as classes *Page Objects* e extrair os dados e métodos encapsulados que representarão as interações entre o usuário e o SUT.

Um dos desafios do teste para aplicações *web* é a compatibilidade da aplicação entre os *browsers*, pois existem vários no mercado e cada um deles se comporta de uma maneira diferente (MYERS et al., 2012). Com o Selenium WebDriver

é possível executar os testes em diversos *browsers*, permitindo obter os resultados dos testes para os mais variados *browsers* do mercado.

A Figura 10 esquematiza a arquitetura da ferramenta proposta por este trabalho. A entrada da ferramenta proposta consiste na URL do SUT e nas classes *Page Objects* da aplicação *web*. A modelagem e processamento é a etapa onde o usuário irá elaborar os modelos e os casos de testes, com o apoio da ferramenta MBTS4MA, concretizará os casos de teste com o Selenium WebDriver e com a biblioteca *JavaParser*, resultando nos códigos de teste como saída da ferramenta.

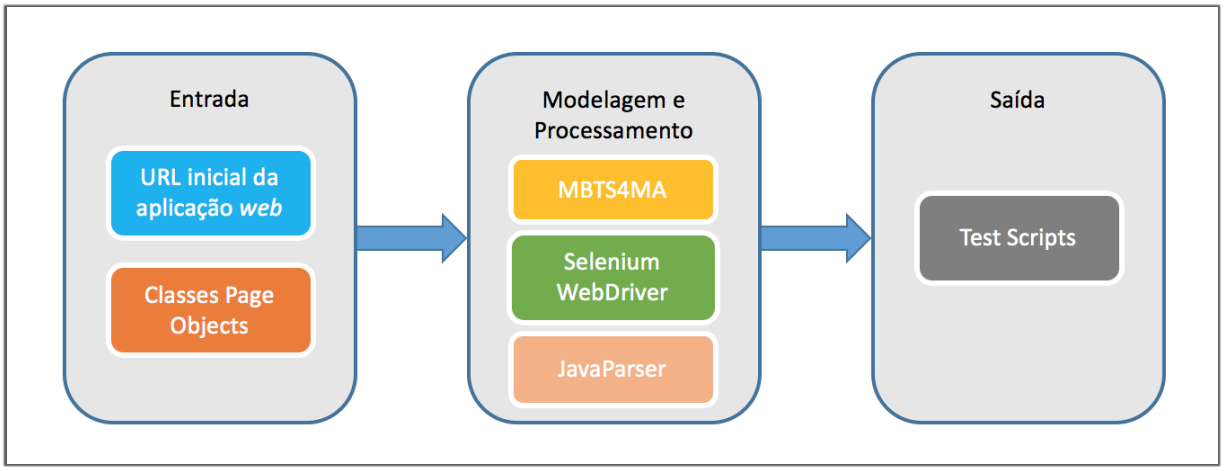

Figura 10: Arquitetura da ferramenta proposta
Fonte: Autoria própria

O cronograma compreende o período de agosto de 2016 a junho de 2017 e está dividido entre as seguintes atividades.

Atividade 1: Definição do tema do trabalho.

Atividade 2: Estudar Selenium WebDriver, *framework* que será utilizado para concretização dos casos de teste.

Atividade 3: Estudar a abordagem de Teste Baseado em Modelo, foco do trabalho.

Atividade 4: Buscar trabalhos relacionados que apresentaram alguma solução para o problema de realizar testes baseados em modelo de forma automática, utilizando o framework Selenium.

Atividade 5: Entrega e apresentação da proposta.

Atividade 6: Realizar as correções na proposta sugeridas pela banca.

Atividade 7: Estudar o funcionamento e estrutura da ferramenta MBTS4MA, que será utilizada na integração com o *framework* Selenium WebDriver.

Atividade 8: Estudar ferramentas que auxiliam na geração automática de classes Page Objects da aplicação *web* que será testada.

Atividade 9: Trabalhar na integração do *framework* Selenium WebDriver e da ferramenta MBTS4MA, artefato que será resultado deste trabalho.

Atividade 10: Condução do estudo de caso que irá validar o artefato gerado pelo trabalho.

Atividade 11: Elaboração do trabalho com apresentação dos resultados obtidos.

Atividade 12: Entrega e apresentação do trabalho final para a banca examinadora.

Atividade 13: Correções no trabalho final conforme sugestões da banca.

A Tabela 1 apresenta o cronograma planejado para a elaboração do trabalho.

ATIVIDADES	Ago 2016	Set 2016	Out 2016	Nov 2016	Dez 2016	Jan 2017	Fev 2017	Mar 2017	Abr 2017	Mai 2017	Jun 2017
1. Definição do tema	■										
2. Estudar Selenium WebDriver		■	■	■	■	■	■	■	■	■	■
3. Estudar Teste Baseado em Modelo		■	■	■	■	■	■	■	■	■	■
4. Buscar trabalhos relacionados		■	■	■	■	■	■	■	■	■	■
5. Entrega e apresentação da proposta			■								
6. Correções na proposta			■	■							
7. Estudar ferramenta MBTS4MA				■	■	■	■	■	■	■	■
8. Estudar ferramentas de apoio ao *Page Object*					■	■	■	■	■	■	■
9. Integrar Selenium WebDriver e ferramenta MBTS4MA						■	■	■	■	■	■
10. Condução do estudo de caso								■	■	■	■
11. Elaboração do trabalho								■	■	■	■
12. Entrega e apresentação do trabalho final										■	■
13. Correções no trabalho final											■

Tabela 1: Cronograma
Fonte: Autoria própria

REFERÊNCIAS

ANAND, Saswat.; BURKE, Edmund K.; CHEN, Tsong. Y.; CLARK, John; COHEN, Myra B.; GRIESKAMP, Wolfgang; HARMAN, Mark; HARROLD, Mary J.; MCMINN, Phil. **An orchestrated survey of methodologies for automated software test case generation**. Journal of Systems and Software, 86(8):1978–2001, Aug. 2013.

BELLI Fevzi; BUDNIK Christof. J.; WHITE Lee. **Event-based modelling, analysis and testing of user interactions:** approach and case study. Software Testing, Verification & Reliability 2006a.

BELLI Fevzi; BUDNIK Christof. J.; SCHIEFERDECKER, Ina. **Test Generation Using Event Sequence Graphs**. Departament of Electrical Engineering, University of Paderborn, 2006b.

BELLI, Fevzi; ENDO, André T.; LINSCHULTE, Michael; SIMAO, Adenilso. **A holistic approach to model-based testing of Web service compositions**. Software – Practice and Experience, 2012.

BINDER, Robert V. **Open Source Tools for Model-Based Testing**. Disponível em: <http://robertvbinder.com/open-source-tools-for-model-based-testing>. Acesso em 07 out. 2016.

CLEAVELAND, Rance; GANESAN, Dharmalingam; GOLDMAN, Daniel; LINDVALL, Mikael; SCHULZE, Christoph. **Assessing model-based testing:** an empirical study conducted in industry. ICSE Companion 2014. Hyderabad, India, p. 135-144, jun. 2014.

DELAMARO, Marcio E.; MALDONADO, José C.; JINO, Mario. **Introdução ao teste de software**. Elsevier Editora Ltda. Rio de Janeiro, 2007. 394 p.

ECMA. **ECMA International**. Disponível em: <http://www.ecma-international.org>. Acesso em: 22 set. 2016.

EMER, Maria C. F. P.; JINO, M.; VERGILIO, Silvia, R. Teste de Aplicações *Web*. In: DELAMARO, Marcio E.; MALDONADO, José C.; JINO, Mario. **Introdução ao teste de software**. Elsevier Editora Ltda. Rio de Janeiro, 2007. p.209-230.

ENDO, André T. **Model based testing of servisse oriented applications**. Tese (Doutorado). USP - São Carlos, SP. 2013.

FARTO, Guilherme de C. **Uma Contribuição ao Teste Baseado em Modelo no Contexto de Aplicações Móveis**. Dissertação (Mestrado). UTFPR - Cornélio Procópio, PR. 2016.

FOWLER, Martin. **PageObject**. Disponível em: <http://martinfowler.com/bliki/PageObject.html>. Acesso em: 22 set. 2016.

GONÇALVES, Edson. **Desenvolvendo Aplicações Web com JSP, SERVLETS, Javaserver Faces, Hibernate, EJB 3 Persistence e Ajax**. Ciência Moderna Ltda. Rio de Janeiro, 2007.

HUANG, Richard. **Selenium Page Object Generator**. Disponível em: <https://github.com/rickypc/selenium-page-object-generator>. Acesso em: 22 set. 2016.

JAVAPARSER. **Java Parser and Abstract Syntax Tree**. Disponível em: <http://javaparser.org>. Acesso em: 22 set. 2016.

LEOTTA, Maurizio; RICCA, Filippo; STOCCO, Andrea; TONELLA, Paolo. **Why Creating Web Page Objects Manually If It Can Be Done Automatically?** 10th International Workshop on Automation of Software Test, IEEE/ACM 2015.

MICSKEI, Zoltán. Model-based tesing (MBT). Disponível em: <http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html>. Acesso em: 07 out. 2016.

MYERS, Glenford. J.; SANDLER, Corey; BADGETT, Tom. **The art of software testing**. John Wiley & Sons. New Jersey, 2011.

ORSO, Alessandro; ROTHERMEL, Gregg. 2014. **Software testing**: a research travelogue (2000–2014). In Proceedings of the on Future of Software Engineering (FOSE 2014). ACM, New York, NY, USA, 117-132. DOI: http://dx.doi.org/10.1145/2593882.2593885

OTWELL, Taylor. **Laravel - The PHP Framework For Web Artisans**. Disponível em: <https://laravel.com>. Acesso em 22 set. 2016.

PRESSMAN, Roger S. **Engenharia De Software**. McGraw Hill, 6a EDIÇÃO. 2006.

ROBOTIUM. Disponível em: <https://github.com/RobotiumTech/robotium>. Acesso em: 22 set. 2016.

SELENIUM - Web Browser Automation. Disponível em: <http://www.seleniumhq.org>. Acesso em: 07 set. 2016.

SINHA, Avik; SMIDTS, Carol. **HOTTest**: A model-based test design technique for enhanced testing of domain-specific applications. ACM Transactions on Software Engineering and Methodology (TOSEM), v. 15, n. 3, p. 242–278, 2006.

THE PHP GROUP. **PHP**: Hypertext Preprocessor. Disponível em: <http://www.php.net>. Acesso em: 22 set. 2016.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL**. Disponível em: <https://www.postgresql.org>. Acesso em: 22 set. 2016.

UTTING, Mark; LEGEARD, Bruno. **Practical model-based testing:** A tools approach. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

UTTING, Mark; PRETSCHNER, Alexander; LEGEARD, Bruno. **A Taxonomy of Model-Based Testing**. Technical Report, Hamilton, New Zealand, 2006.

W3C. **HTTP - Hypertext Transfer Protocol**. Disponível em: <https://www.w3.org/Protocols>. Acesso em: 15 set. 2016.

W3C. **W3C Cascading Style Sheet**. Disponível em: <https://www.w3.org/Style/CSS/>. Acesso em: 22 set. 2016.

W3C. **W3C HTML**. Disponível em: <http://www.w3.org/html>. Acesso em: 22 set. 2016.

WALES, M. **Web Dev Careers Decoded**: Front-End vs Back-End vs Full Stack. Disponível em: <http://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html>. Acesso em: 22 set. 2016.

ZIVANOVIC, Dejan. **Test Automation in Selenium Using Page Object Model and Page Factory.** Disponível em: < https://www.toptal.com/selenium/test-automation-in-selenium-using-page-object-model-and-page-factory>. Acesso em 22 set. 2016.