STEPHAN BARROS SUMI

UMA ABORDAGEM DE TESTES DE UNIDADE AUTOMATIZADOS PARA APLICAÇÕES WEB LEGADO

Trabalho apresentado como requisito parcial a obtenção do grau de Bacharel em Ciência da Computação no curso de graduação em Ciência da Computação, Setor de Exatas da Universidade Federal Do Paraná.
Orientador: Prof. Bruno Müller Junior

Curitiba

2017

Sumário

1 Introdução

Como um dos principais atores do século 21, a Internet e seus usuários vêm se expandindo rapidamente desde sua abertura para as massas, na década de 90. Em 2016, estimou-se que cerca de 47% das pessoas do mundo utilizavam a Internet [5], possibilitando novas formas de interatividade social, atividades, entretenimento, negócios e telecomunicação, entre outros.

De seu início até hoje, a Internet deixou de ser somente um repositório de páginas estáticas e passou a ser utilizada, também, dinamicamente, como um veículo de desenvolvimento e disponibilização dos mais diversos tipos de aplicações [21].

Aplicações tradicionais de *desktops*, geralmente apresentam limitações como funcionamento em sistemas operacionais específicos, equipamentos no qual operam, e necessidade de atualizações individuais em cada cliente.

Por outro lado, aplicações web oferecem uma maior simplicidade em seu desenvolvimento. Usuários acessam a aplicação através do navegador web de sua preferência, e trabalham com os recursos disponibilizados através da internet. Atualizações são aplicadas somente no servidor, e não se faz mais necessário reproduzir o processo em cada usuário [20].

Entretanto, esta nova abordagem nos força a reavaliação de vários conceitos e métodos do desenvolvimento de software para adequá-los a nova realidade das aplicações web, dos quais destacam-se os métodos de teste.

Métodos tradicionais de teste de software assumem em geral a existência de dois elementos somente, a aplicação e o banco de dados. Aplicações web, entretanto, adicionam mais um elemento, o navegador web. Enquanto alguns *frameworks* de desenvolvimento mais recentes já incluem métodos para abordagem dos testes e automatização destes, devido ao rápido crescimento da web e suas tecnologias, aplicações web antecessoras sofrem com a complexidade de realizar os testes de software necessários, dificultando sua manutenção e atualização

Este trabalho descreve a criação de uma ferramenta automatizada de teste de unidade de aplicações web legado, sem a necessidade de refatoração de código existente ou migração para *frameworks* com este tipo de suporte.

2 Fundamentos

Este capítulo descreve os conceitos importantes que serão utilizados neste trabalho. A Seção 2.1 apresenta uma breve explicação sobre aplicações web e sua história. Na Seção 2.2 é abordado o teste de software de aplicações web, estratégias de teste e ferramentas existentes. Finalmente, na seção 2.3 é detalhado o objetivo do trabalho.

2.1 Aplicações Web

Aplicações web são, de forma geral, programas de computador que são utilizados através de um navegador web para executar seu propósito. Alguns exemplos mais comuns desse tipo de aplicação são *webmail*, sites de venda e chats, entre outros.

Estas aplicações seguem a estrutura de modelo cliente-servidor, a qual segrega as tarefas de requisitantes de conteúdo para os clientes e o de provedores de conteúdo para os servidores [23]. Em aplicações web, os clientes são os navegadores web, como o *Mozilla Firefox*, *Google Chrome* e *Microsoft Internet Explorer*. Já os servidores são os servidores web, como o *Apache HTTP Server*, *Internet Information Services* e *NGINX*. Cliente e servidor comunicam-se através do protocolo HTTP ou HTTPS. A Figura 1 ilustra a arquitetura geral de aplicações que seguem o modelo cliente-servidor.

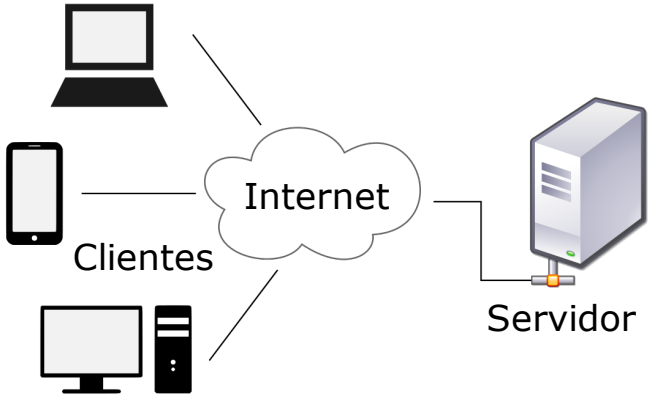

Figura 1: Arquitetura de aplicações cliente-servidor[25].

As aplicações web são um marco na evolução da *World Wide Web*. Introduzida no início da década de 90, a *World Wide Web* nasceu com o intuito de tornar possível o acesso a informação de forma consistente e simples. Utilizando-se de hipertexto e do protocolo HTTP, tornou-se possível compartilhar facilmente documentos e informações de servidores para clientes. Mesmo contando com recursos para criação de formulários [4] desde suas versões iniciais, a grande maioria do conteúdo disponível era estático.

Uma das primeiras mudanças neste cenário estático ocorreu em 1995, quando a *Netscape* lançou, junto com seu navegador *Netscape Navigator 2.0*, o *JavaScript*, linguagem de programação interpretada com o intuito de executar *scripts* do lado do cliente, possibilitando que programadores adicionassem um certo nível de dinamicidade às páginas web, sem a necessidade de interação com o servidor. No ano seguinte, a *Macromedia* introduz o *Flash*, reprodutor de animações que podia ser integrado aos navegadores.

Somente em 1999 a palavra aplicação web foi introduzida. Ela apareceu nas especificações do *Java Servlet API* versão 2.2, a qual adicionou suporte às aplicações web na linguagem de programação Java [6].

E finalmente, em 2005, surgiu o termo *Ajax*, para o envio e carregamento assíncrono de informações com o servidor. Apesar de operações assíncronas já estarem disponíveis anteriormente desde 1999, o termo *Ajax* foi consagrado somente neste ano, por *Jesse James Garret* em seu artigo *Ajax: A New Approach to Web Applications* [1]. No ano seguinte, o *World Wide Web Consortium*, principal organização de padronização da *World Wide Web*, lançou a primeira especificação do objeto *XMLHttpRequest*, formalizando este tipo de operação[16].

2.2 Teste de software de aplicações web

O teste de aplicações web compartilha o mesmo objetivo que o teste de outras aplicações mais tradicionais, entretanto as características distintas deste tipo de aplicação impedem o uso direto das técnicas tradicionais de testes [19]. Entre essas características distintas, encontram-se o número grande de clientes concorrentes, o ambiente heterogêneo de execução, a dinamicidade de execução do programa, que varia de acordo com o usuário e estado do servidor, e a natureza assíncrona das aplicações [24].

Além disso, dois problemas comuns no teste de software se referem a providenciar os valores de entrada à aplicação e observar o comportamento desta. Denominados, respectivamente, de controlabilidade e observabilidade, estas definem o nível de dificuldade em executar as tarefas de teste [18]. Por empregarem um protocolo de comunicação sem estado (HTTP), e possuírem natureza distribuída, na qual a interface com o usuário e servidor estão separados, as aplicações web possuem baixa controlabilidade e observabilidade [20].

O teste de software consiste na avaliação da aplicação para determinar se esta satisfaz os requisitos especificados, através da execução da aplicação utilizando uma combinação de entradas e estados para revelar falhas. Os requisitos de uma aplicação podem ser divididos em duas categorias: requisitos não funcionais e requisitos funcionais [18].

Requisitos não funcionais descrevem como um sistema deve funcionar, e tratam de atributos de qualidade. Testes de requisitos não funcionais, portanto, visam avaliar o sistema em quesitos de desempenho, compatibilidade, segurança, usabilidade e acessibilidade [19]. Para a verificação de cada um dos requisitos desta categoria, é necessário conceber atividades de teste específicas. A Tabela 1 lista as principais categorias de requisitos não funcionais e as atividades que podem ser realizadas nos testes.

Os requisitos funcionais, por outro lado, definem as funções e tarefas o que o sistema deve realizar. Logo, testes de requisitos funcionais visam descobrir erros vindos de falhas na implementação [19]. Para este fim, a abordagem tradicional de teste de software ainda se aplica, na qual podemos definir o processo de teste como um conjunto de atividades de teste realizadas considerando-se diferentes níveis: teste de unidade, teste de integração e teste de sistema [17].

O teste de unidade tem por objetivo a verificação individual de cada componente do sistema. O teste de integração considera vários componentes em conjunto em seus testes, a fim averiguar seu correto funcionamento. Por último, o teste de sistema abrange todos os componentes do sistema, a fim de verificar se o mesmo está de acordo com os requisitos como um todo [18].

As subseções abaixo descrevem mais a fundo os níveis de teste, bem como as estratégias disponíveis para elaboração e abordagem dos testes.

2.2.1 Testes de unidade, integração e sistema

O teste de aplicações web é similar ao teste de páginas web. Do lado do cliente, as páginas web são responsáveis pela exibição do conteúdo, recebimento de entrada de informações por parte do usuário e navegação pela aplicação. Do lado do servidor, as páginas web são responsáveis pela validação da lógica da aplicação, armazenamento e consulta de dados [19].

Os testes de unidade podem ser realizados através de técnicas de caixa preta,

Categoria	Descrição
Desempenho	Desempenho do sistema em quesitos como tempo de resposta e disponibilidade. Os testes nesta categoria são feitos, em geral, com a simulação de múltiplos acessos simultâneos ao sistema por um determinado intervalo de tempo.
Compatibilidade	Refere-se ao ambiente no qual a aplicação deve funcionar. Os testes deverão revelar falhar oriundas do uso de diferentes plataformas de servidores web e navegadores de internet.
Segurança	Diz respeito a efetividade da aplicação em impedir o acesso à usuários não autorizados, garantir o acesso à recursos e funcionalidades para usuários autorizados, e preservar os recursos do sistema contra o mal-uso.
Usabilidade	Concerne ao quão fácil a aplicação é de ser utilizada, abordando questões de interface do usuário, incluindo a renderização de gráficos e textos e a clareza de mensagens e comandos exibidos.
Acessibilidade	Inclui-se aqui testes que tem como objetivo verificar a funcionalidade da aplicação em situações de redução de configurações mínimas de hardware e/ou software do lado do cliente, como bloqueio de execução de scripts ou gráficos, bem como o uso da aplicação por pessoas portadoras de deficiências.

Tabela 1: Tipos de requisitos não funcionais.

branca ou cinza [18]. Do lado do cliente, em geral, os testes irão detectar erros de conteúdo exibido na página, links funcionando de forma incorreta, erros na execução de itens interativos na página (como botões), e erros na execução de scripts. Já do lado do servidor, usualmente serão identificados erros na inserção e consulta de dados, na execução do servidor web, e erros na geração de páginas dinâmicas.

Testes de integração compreende em testar agrupamentos de páginas web relacionadas, a fim de verificar o funcionamento em conjunto destas. Páginas web podem estar relacionadas através de links ou redirecionamentos, ou pela sua participação na implementação de algum requisito funcional. A fim de realizar os testes, conhecimentos da estrutura e comportamento da aplicação são necessários, portanto estratégias de caixa cinza são consideradas mais adequadas nesta etapa [19].

Por último, os testes de sistema visam validar o sistema como um todo perante seus requisitos, encontrando erros que dependam de toda a aplicação. Para isso, é comum o uso de técnicas de caixa preta ou cinza [18].

2.2.2 Estratégias de teste

As estratégias de teste definem como serão abordadas a criação dos casos de teste. Elas são divididas em três categorias: caixa preta, caixa branca e caixa cinza. Testes de caixa preta são elaborados com base nas funcionalidades esperadas do sistema. Já os testes de caixa branca efetuam a análise do código fonte da aplicação para desenvolvimento dos casos de teste, por último, os tes-

tes de caixa cinza são elaborados utilizando uma abordagem dupla, combinando aspectos de caixa branca e preta [18][19].

2.2.3 Ferramentas para teste de aplicações web

Ferramentas para teste de software podem auxiliar as atividades de teste com a automatização de algum dos processos de teste. Em geral, elas irão automatizar a geração de casos de teste, a execução dos casos de teste ou a validação dos resultados do teste.

Para aplicações web, a grande maioria das ferramentas existente são projetadas para efetuar testes de desempenho e segurança. Poucas são projetadas para realizar testes de requisitos funcionais[19].

Aplicativos como o Sandbox [12], Mocky [8] e Postman [10], restringem-se a testar apenas web services, aplicações desenvolvidas para comunicação entre máquinas. Portanto, não são ferramentas ideais para testar aplicações web ou websites em geral.

O Selenium IDE [13] é uma ferramenta com o propósito de testar aplicações web. Operando como uma extensão para o navegador web *Mozilla Firefox*, ele grava a interação do usuário com uma página web qualquer, gerando um script das ações realizadas e possibilitando a reprodução delas.

Apesar de realizar o que se propõe, a dependência com o navegador *Mozilla Firefox* limita a utilidade da ferramenta, e dificulta a realização de testes extensos.

Outras ferramentas como o EveryStep Scripting Tool [3] geram testes gravando as ações do usuário no website, simulando cliques de mouse e teclas pressionadas no teclado. Como essas ações são gravadas de forma absoluta, alterações na localização de qualquer elemento no website resultam na reavaliação de todos os testes anteriormente preparados.

2.3 Objetivos

O teste de software é uma parte fundamental do desenvolvimento de qualquer aplicação, e está presente independentemente do modelo de desenvolvimento escolhido. Portanto, além de cumprir de forma eficaz seu papel, quando se trata de aplicações web, buscamos também soluções que sejam flexíveis e se adaptem à natureza heterogênea da internet.

Nosso objetivo então é criar uma solução de testes que se adapte às diversas tecnologias existentes, que possa vir a ser executada de forma distribuída e que elimine elementos desnecessários ao processo de teste. Como há abundância de ferramentas para testes de requisitos não funcionais, este trabalho irá focar no teste de requisitos funcionais.

3 Descrição conceitual do modelo

Este capitulo descreve conceitualmente o processo proposto para a realização de teste de unidade em aplicações web.

Independentemente dos métodos e ferramentas de desenvolvimento utilizados, o teste de software consiste em cinco atividades, ilustradas na Figura 2. Enquanto essas atividades não ocorrem necessariamente nesta ordem, essas etapas sempre estão presentes no ciclo de teste de software.

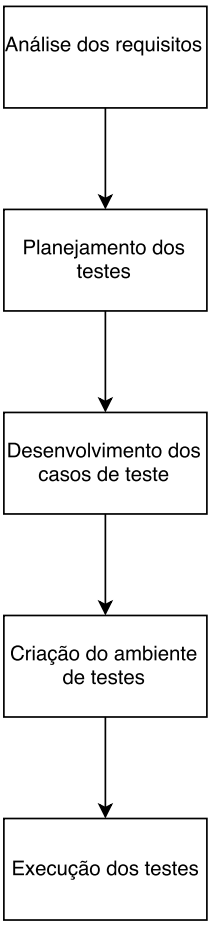

Figura 2: Atividades de teste de software.

Enquanto algumas arquiteturas de software, como o padrão MVC (*Model view controller*) e *frameworks* de desenvolvimento podem facilitar a realização de testes, aplicações legados que não foram desenvolvidas com a utilização de tais recursos sofrem com o processo de realização de testes.

Devido à natureza da atividade, as etapas de planejamento dos testes e desenvolvimento dos casos de teste resultam muitas vezes na execução exaustiva de elementos do software. Em aplicações web, isso se traduz na inserção de dados, consulta e interação com elementos de uma página web, como formulários e links.

Portanto, o objetivo deste trabalho consiste na criação de uma aplicação para a realização automatizada da etapa de execução dos testes. O automatizador irá realizar automaticamente a verificação de aplicações web, preenchimento de dados e consulta dos resultados, a partir de documentos de execução de testes fornecidos.

Para realizar essa atividade, além do desenvolvimento da aplicação de automatização, faz-se necessário também alterar outra atividade de teste de software, a de desenvolvimento dos casos de teste. Durante essa etapa, na qual são ge-

rados os casos de teste, é necessário padronizar o documento gerado (ou criar um novo artefato a partir dele) para que este se torne compreensível para a aplicação de automatização. A esse arquivo contendo os casos de teste em um formato acessível, denominaremos o nome *documento de execução*.

A Figura 3 ilustra, baseando-se na Figura 2 mostrada anteriormente, as etapas de teste com as quais iremos interagir neste trabalho.

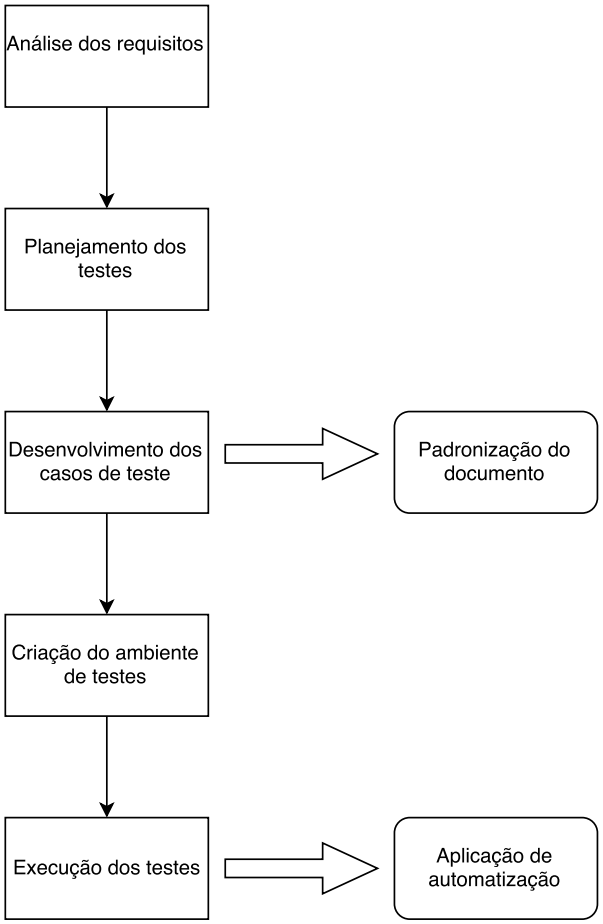

Figura 3: Atividades deste trabalho enquadradas nas etapas de teste de software.

As seções seguintes aprofundam cada etapa do processo de desenvolvimento desta solução. Na Seção 3.1 será abordada a criação de casos de teste. Na Secão 3.2 será discutido sobre o documento de execução que será utilizado pela aplicação para efetuar os testes. Na Seção 3.3 será apresentada a aplicação de automatização que irá efetuar a leitura dos documentos de execução e realizar os testes.

3.1 Casos de teste

Caso de teste é um conjunto de condições, procedimentos e dados de entrada e saída que detalham, passo a passo, como realizar o teste de algum recurso em

Titulo	Um nome exclusivo utilizado para identificar o caso de teste.
URL	URL da página com a aplicação web que está sendo testada.
Procedimento	Uma descrição passo a passo para execução do teste.
Dados de entrada	Valores de entrada necessários para executar as ações no sistema.
Resultados esperados	Qual o resultado esperado para cada valor de entrada.

Tabela 2: Documento de caso de teste.

uma aplicação, a fim de validar seu correto funcionamento.

Os casos podem ser descritos em documentos padronizados, contendo todas as informações necessárias para a realização do teste. Para o desenvolvimento desta solução iremos utilizar uma versão simplificada, com os campos titulo, procedimento, dados de entrada, resultados esperados e URL alvo. A Tabela 2 ilustra o documento proposto e a descrição de cada campo.

3.2 Documento de execução

Com a escrita dos casos de teste, identificam-se elementos com os quais a aplicação de automatização de testes precisará interagir. O documento de execução, portanto, será a ponte entre os casos de teste e a aplicação de automatização.

O documento de execução irá conter todos os dados do caso de teste, acrescidos de identificadores dos elementos web com os quais o teste interage, e as ações a serem realizadas em cada elemento.

Neste trabalho, os identificadores dos elementos web serão os *ids* dos elementos HTML que formam os componentes da aplicação web, como formulários, hiperlinks e botões, que estão sendo testados.

A Lista 1 ilustra um exemplo de código HTML, no qual, supondo um caso de teste de formulários, podemos identificar os elementos de interesse como sendo as tags *input*, que possibilitam a inserção dos valores de entrada e envio do formulário.

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4    <form action="exemplo.php">
5      Primeiro campo:
6      <input id="campo1" type="text" name="c1" value="">
7      Segundo campo:
8      <input id="campo2" type="number" name="c2" value="">
9      <input id="botaoEnvio" type="submit" value="Submit">
10   </form>
11 </body>
12 </html>
```

Lista 1: Página HTML de exemplo a ser testada.

Para tornar este documento de execução compreensível para a aplicação de automatização, optaremos pela escrita do documento no formato JSON (JavaS-

Titulo	Teste de formulário
URL	http://localhost/
Procedimento	Navegar para a página Inserir os valores de entrada no formulário Pressionar o botão Enviar

Dados de entrada	Resultados esperados
[ABC, 123]	Redireciona para a página exemplo.php
[AAA, AAA]	Exibe janela de erro

Tabela 3: Exemplo de caso de teste preenchido.

cript Object Notation) pela sua característica facilidade de escrita e compreensão por pessoas, porém mantendo-se de fácil análise por programas [7][22].

O documento de execução consistirá em um arquivo JSON contendo o título do caso do teste, a URL da aplicação web e o procedimento de execução, composto pelo identificador dos elementos HTML de interesse, a ação executada em cada um deles, e os valores de entrada e saída, se aplicáveis.

Cada caso de teste irá gerar um documento de execução JSON, contendo os dados citados. Abaixo, é detalhado o nome de cada chave e uma breve descrição de sua funcionalidade.

- Titulo: Título do teste conforme descrito no caso de teste.
- URL: URL da aplicação.
- IdElemento: Um vetor com os identificadores dos elementos HTML que serão utilizados no teste, em ordem de interação.
- DadosEntrada: Uma lista que contém os dados de entrada do teste. Possui as subchaves:
  - Acao: Vetor de ações a serem realizadas nos elementos da chave IdElemento, em ordem de execução.
  - Valor: Vetor de valores a serem utilizados nos elementos pelas ações descritas. Pode ser nulo, conforme necessidade da ação.
- ResultadosEsperados: Vetor que contém os resultados esperados para cada dado de entrada. Cada elemento deste vetor possui as subchaves:
  - Acao: Resultado esperado após a execução do procedimento de teste.
  - Valor: Valor esperado após a execução dos procedimento de teste. Pode ser nulo.

Considere o trecho HTML da Lista 1 e o caso de teste especificado na Tabela 3. O documento de execução JSON resultante está descrito na lista 2. As flechas em vermelho (↪) indicam uma quebra de linha visual, para facilitar a leitura do documento.

```
1 {
2   "Titulo":"Teste de formulario",
3   "URL":"http://localhost",
```

```
4     "IdElemento": ["campo1", "campo2", "botaoEnvio"],
5     "DadosEntrada":{
6         "Acao":["InserirValor", "InserirValor", "Clique"
              ↪ ],
7         "Valor":[ ["ABC","123", null], ["AAA","AAA",
              ↪ null] ]
8     },
9     "ResultadosEsperados":[
10      {
11        "Acao":"Redirecionamento",
12        "Valor":"http://localhost/exemplo.php"
13      },
14      {
15        "Acao":"Alerta",
16        "Valor":null
17      }
18    ]
19  }
```

Lista 2: Documento de execução JSON.

3.3 Aplicação de automatização

A aplicação de automatização será responsável pela execução dos testes na aplicação web. Seu fluxo de execução consistirá na leitura do documento de execução, interpretação dos passos e instruções descritos nele, e execução dos testes.

Devido ao formato escolhido para o documento de execução, a leitura deste é efetuada de forma simplificada - várias linguagens de programação já possuem, nativamente, bibliotecas que executam a análise de documentos JSON.

Em seguida, deveremos estabelecer formas de conexão com a aplicação web e determinar os métodos de acesso e interatividade com seus elementos. Será necessário criar métodos que reflitam todas as interações especificadas no campo *Acao* dos documentos de execução, como cliques, inserção de dados e consulta a elementos da página.

Finalmente, após a execução dos testes, a aplicação deverá gerar um relatório, contendo detalhes das atividades realizadas, como a aprovação ou reprovação da aplicação web em cada caso de teste testado.

4 Conceitos e detalhes de implementação

Este capítulo descreve os detalhes de implementação do processo definido para a realização de testes de unidade automatizados.

Para realizar a implementação da aplicação de automatização de testes definida anteriormente, utilizaremos a linguagem de programação Python, pela sua gama de recursos *open-source* disponíveis, incluindo recursos a serem utilizados nesta implementação a serem detalhados posteriormente neste capítulo, além da grande quantidade de documentação disponível na web. Os algoritmos apresentados neste texto, todavia, serão exibidos em pseudocódigo, a fim de facilitar sua compreensão.

Definida a linguagem de programação, necessitamos determinar a forma com a qual iremos interagir com as aplicações web. Soluções mais comuns para processamento de páginas web como *BeautifulSoup* [2] e *urllib2* [11] não atendem ás necessidades desta aplicação, pois coletam apenas o código HTML das páginas fornecidas, excluindo portanto qualquer processamento que pode vir a ocorrer do lado do cliente por *scripts*, como carregamentos assíncronos de informações, além de impossibilitarem a interação com formulários e outros elementos dinâmicos.

Sendo assim, para os fins desta aplicação utilizaremos a biblioteca *Selenium* do projeto *Selenium WebDriver* [14]. Esta biblioteca dispõe de métodos para realizar a condução de navegadores web de forma programática, oferecendo suporte para operações de navegação e interatividade. O Selenium dispõe de suporte ao navegadores *Google Chrome*, *Mozilla Firefox*, *Microsoft Internet Explorer* e *PhantomJS*, entre outros.

A fim de atender os objetivos determinados anteriormente, optou-se pela escolha do PhantomJS [9] como navegador a ser utilizado pelo Selenium. Esta escolha deve-se ao fato de o navegador incluir suporte a vários padrões da web, como o modelo de objeto de documentos, *CSS*, *Canvas*, e também a execução de *JavaScript*. Além disso, o PhantomJS não realiza a renderização das páginas, tornando sua navegação mais rápida que outros navegadores, e ideal para o cenário de automatização.

Com o intuito de ilustrar o desenvolvimento da aplicação e determinar as funcionalidades a serem implementadas, iremos utilizar uma aplicação web simplificada como alvo dos testes, ilustrada na figura 4.

Esta aplicação realiza o cadastro de pessoas com os campos nome e número. Ambos os campos precisam ser únicos. Pessoas já cadastradas são exibidas em uma tabela, que é atualizada automaticamente caso haja um novo cadastro. Ao selecionar um elemento da tabela, é exibido um botão que permite a remoção do elemento selecionado.

O código exibido na Lista 3 contém os trechos HTML de interesse para a realização do teste. As linhas 1 a 7 contêm o formulário utilizado durante um novo cadastro de pessoa. A linha 8 exibe o código do botão de remoção de cadastro. As linhas 9 a 13 exibem o primeiro elemento da tabela de pessoas já cadastradas.

```
1  <form name="insertForm" ng-submit="inserePessoa()">
2      <label>Nome</label>
3      <input type="text" id="formulario_nome">
4      <label>Numero</label>
5      <input type="number" id="formulario_numero">
6      <md-button type="submit" id="formulario_enviar">
           ↪ Cadastrar</md-button>
7  </form>
8  <button ng-click="deletaPessoa()" id="botao_delete">
       ↪ Deletar selecionado</button>
9  <tr id="tr_100">
10     <td><md-checkbox></md-checkbox></td>
11     <td md-cell="" class="md-cell ng-binding">Sylvia
           ↪ Vaughn</td>
12     <td md-cell="" class="md-cell ng-binding md-numeric">
```

```
        ↪ 100</td>
13 </tr>
```

Lista 3: Trecho de código HTML da aplicação de demonstração.

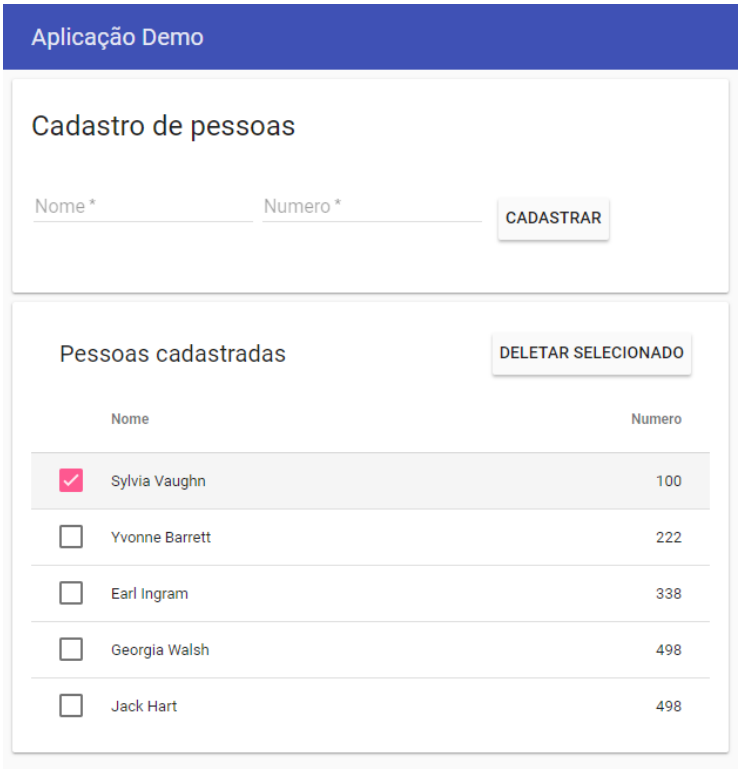

Figura 4: Aplicação de demonstração.

Para testar o automatizador, definiremos dois testes sobre a aplicação web e seus respectivos documentos de execução. Os testes irão avaliar as funcionalidades de cadastro e remoção de pessoas.

O primeiro teste, cujo documento de execução é exibido na Lista 4, intitulado *Teste de cadastro de pessoas*, efetua o teste inserção de dois cadastros na aplicação. O primeiro, válido, deverá ser cadastrado com sucesso. Já o segundo é inválido, e não deverá ser aceito pela aplicação. Para verificação do resultado, serão implementadas duas funções de verificação, denominadas *ElementoExisteXPath* e *ElementoNaoExisteXPath*. Essas funções verificam se há algum elemento na tabela de pessoas cadastradas que contém os dados inseridos no cadastro.

XPath neste caso, refere-se à linguagem de consulta em documentos XML, frequentemente utilizada em aplicações de automatização web pela sua capacidade de localizar elementos HTML através da sua estrutura, quando não há outros identificadores disponíveis [15].

```
1 {
2   "Titulo":"Teste de cadastro de pessoas",
```

```
  "URL":"website/index.html",
  "IdElemento":["formulario_nome", "formulario_numero"
    ↪ , "formulario_enviar"],
  "DadosEntrada":{
    "Acao":["InserirValor", "InserirValor", "Clique"],
    "Valor":[ ["Joao Sergio","123", null], ["Maria
      ↪ Joana","Pedro", null] ]
  },
  "ResultadosEsperados":[
    {
      "Acao":"ElementoExisteXPath",
      "Valor":"//td[contains(text(), 'Joao Sergio')]/
        ↪ following-sibling::td[contains(text(), '
        ↪ 123')]"
    },{
      "Acao":"ElementoNaoExisteXPath",
      "Valor":"//td[contains(text(), 'Maria Joana')]/
        ↪ following-sibling::td[contains(text(), '
        ↪ Pedro')]"
    }
  ]
}
```

Lista 4: Documentos de execução de teste de cadastro da aplicação de demonstração.

O segundo caso de teste, denominado *Teste de remoção de pessoas*, efetua o teste de remoção do cadastro do primeiro elemento da tabela de pessoas já cadastradas. Este teste também utiliza-se da função *ElementoNaoExisteXPath* para verificar se o elemento foi realmente removido. Seu documento de execução está disponível na Lista 5.

```
{
  "Titulo":"Teste de remocao de pessoas",
  "URL":"website/index.html",
  "IdElemento":["tr_100", "botao_delete"],
  "DadosEntrada":{
    "Acao":["Clique", "Clique"],
    "Valor":[[null, null]]
  },
  "ResultadosEsperados":[
    {
      "Acao":"ElementoNaoExisteXPath",
      "Valor":"//td[contains(text(), 'Sylvia Vaughn')
        ↪ ]/following-sibling::td[contains(text(), '
        ↪ 100')]"
    }
  ]
}
```

Lista 5: Documentos de execução de teste de remoção da aplicação de

demonstração.

As seções seguintes detalham a implementação individual dos elementos que compõem a aplicação de automatização. A Seção 4.1 detalha o fluxo principal do automatizador. A Seção 4.2 detalha a leitura e validação do documento de execução JSON. A Seção 4.3 detalha a implementação dos métodos de acesso e interação com a aplicação web a ser testada. Por último, a Seção 4.4 concretiza a implementação desta solução, exibindo a execução do automatizador sobre a aplicação de demonstração.

4.1 Automatizador

O automatizador é a parte principal da aplicação. Responsável por receber os dados de entrada e orquestrar o fluxo de execução da aplicação até a sua saída.

A Lista 6 exibe em pseudocódigo a função principal do automatizador.

```
1 FuncaoPrincipal(){
2         processarArgumentos()
3         para cada documento de entrada, faça{
4                 lerDocumento()
5                 validarDocumento()
6                 executarTestes()
7         }
8         imprimirResultados()
9 }
```

Lista 6: Pseudocódigo do automatizador.

A primeira tarefa executada é o processamento dos parâmetros fornecidos durante a execução da aplicação, como os arquivos de entrada e saída. Mais detalhes sobre os parâmetros aceitos são fornecidos no Apêndice A.

Em seguida, é feita a tratativa de cada arquivo fornecido como documento de execução nos parâmetros de entrada. Cada documento é lido e validado como um documento de execução nos parâmetros definidos neste trabalho. Na sequência, são executados os testes determinados pelo documento.

Por último, após a realização de cada teste, é impresso o resultado, indicando se a aplicação foi aprovada ou não nos testes descritos.

4.2 Documento de Execução

A primeira tarefa da aplicação de automatização de testes é efetuar a leitura do Documento de Execução. A aplicação deverá interpretar todas as chaves e valores contidos no documento e criar uma estrutura de acesso para prosseguir com os testes.

Como foi definida a escrita do arquivo no formato JSON, a leitura deste se resume à importação da biblioteca nativa da linguagem escolhida e o uso da função de leitura. A Lista 7 exemplifica essa operação, na qual a função *lerJson(nomeArquivo)* representa o uso da função de leitura nativa da linguagem, que por sua vez retorna um vetor associativo com os dados do arquivo.

```
1 lerDocumento(nomeArquivo){
```

```
2           documento = lerJson(nomeArquivo)
3           retorna(documento)
4   }
```

Lista 7: Pseudocódigo de leitura de arquivo JSON.

Por sua vez, para efetuar a verificação da validade do documento lido, constatando-se se todas as chaves do documento de execução estão presentes conforme definido neste trabalho, dispomos da função *validarDocumento()*, apresentada na Lista 8, a qual retorna verdadeiro para um documento válido e falso caso contrário.

```
1   validarDocumento(documento){
2           chaves = ["Titulo", "URL", "IdElemento", "
                ↪ DadosEntrada", "ResultadosEsperados"]
3           para cada chave:
4                   se chave não existem em documento:
5                           retorna(Falso)
6           subchaves = ["Acao", "Valor"]
7           para cada subchave
8                   se chave nao existe em documento["
                        ↪ DadosEntrada"]
9                           retorna(Falso)
10          para cada valor em documento["ResultadosEsperados
                ↪ "]:
11                  para cada subchave
12                          se a chave nao existe em valor
13                                  retorna(Falso)
14          retorna(Verdadeiro)
15  }
```

Lista 8: Pseudocódigo de validação de arquivo JSON.

4.3 Métodos de acesso e verificação

Nos métodos de acesso e verificação teremos as principais funcionalidades da aplicação de automatização. Este módulo é responsável pela implementação da execução dos testes, incluindo métodos de acesso à aplicação testada, interatividade com a mesma e validação dos testes executados.

A Lista 9 demonstra a estrutura principal da execução dos testes, utilizando o Selenium WebDriver em conjunto com o navegador PhantomJS.

```
1   executarTestes(documento){
2           driver = WebDriver(PhantomJs)
3           para cada teste em documento:
4                   para cada acao em teste:
5                           enfileraAcao()
6                   executaAcoesEnfileiradas()
7                   verificaResultado()
8   }
```

Lista 9: Pseudocódigo da execução de testes.

Para o correto funcionamento desse processo, é necessário implementar, para cada ação definida nos documentos de execução, a função de ação correspondente. Seguindo o exemplo definido no início deste capítulo, as funções a serem implementadas são: *InserirValor*, *Clique*, *ElementoExisteXPath* e *ElementoNaoExisteXPath*.

A implementação dessas funções utiliza recursos disponíveis pelo Selenium Webdriver, conforme demonstrado na lista 10.

```
1  InserirValor(driver, idElemento, valor){
2          elemento = encontraElementoPeloId(idElemento)
3          driver.enviaValoresParaElemento(elemento, valor)
4  }
5  Clique(driver, idElemento){
6          elemento = encontraElementoPeloId(idElemento)
7          driver.clique(elemento)
8  }
9  ElementoExisteXPath(driver, valor){
10         elemento = encontraElementoPeloXPath(valor)
11         se elemento é válido:
12                 retorna(Verdadeiro)
13         senão:
14                 retorna(Falso)
15 }
16 ElementoNaoExisteXPath(driver, valor){
17         retorna(!ElementoExisteXPath(driver, valor))
18 }
```

Lista 10: Pseudocódigo das funções de ação.

4.4 Execução do programa

Concluída a implementação das funções de leitura e validação dos documentos de execução e dos métodos de interação com a aplicação a ser testada, iremos agora efetuar a execução do automatizador sobre a aplicação de demonstração apresentada no início deste capítulo.

A aplicação será executada tendo como entrada os dois documentos de execução definidos anteriormente, nas Listas 4 e 5, nomeados *documentoExecucaoCadastro.json* e *documentoExecucaoRemocao.json*, respectivamente. A Figura 5 ilustra a execução do programa.

Na Figura 5 observamos que todos os testes foram realizados com sucesso e a aplicação de demonstração foi aprovada em todos os testes determinados nos documentos de execução.

O primeiro teste realiza o cadastro de pessoas na aplicação teste. Possuindo dois valores de entrada definidos, o primeiro tenta realizar o cadastro da pessoa de nome *Joao Sergio* e número *123*, ou seja, um cadastro válido. A aplicação deve aceitar essa entrada. A Figura 6 ilustra a captura da tela antes (lado esquerdo) e depois (lado direito) do teste. É possível observar que o cadastro foi realizado com sucesso e é exibido na tabela da aplicação.

Já a segunda entrada do primeiro teste tenta realizar o cadastro da pessoa de nome *Maria Joana* e número *Pedro*, ou seja, um cadastro inválido, devido ao

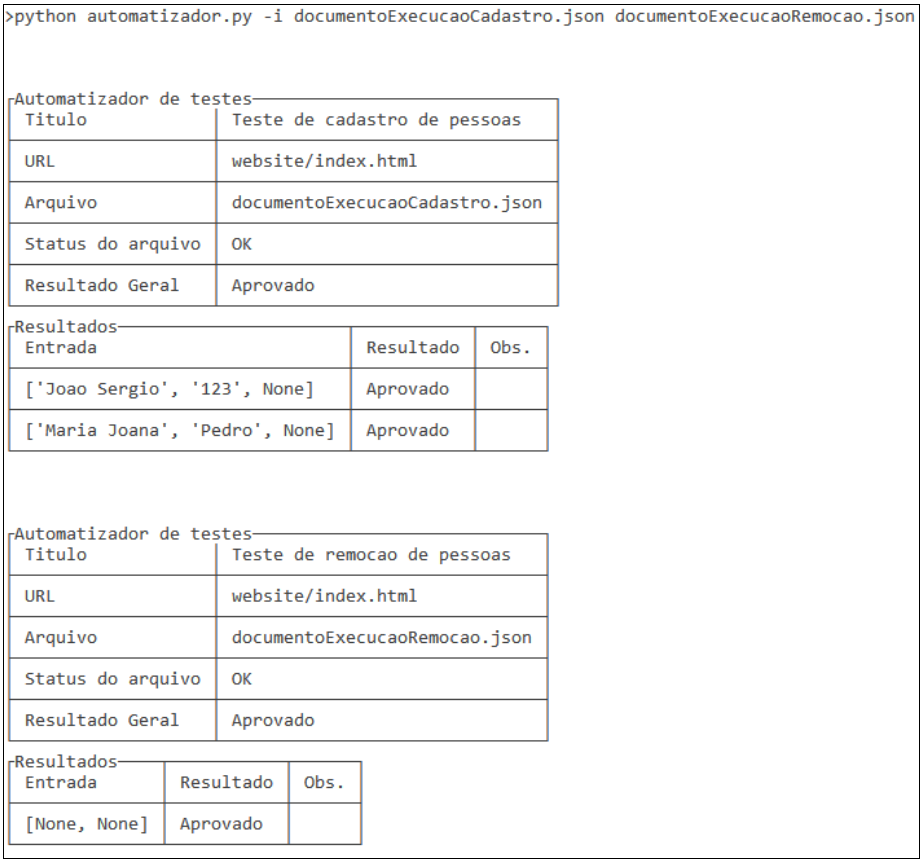

Figura 5: Resultado da execução do automatizador.

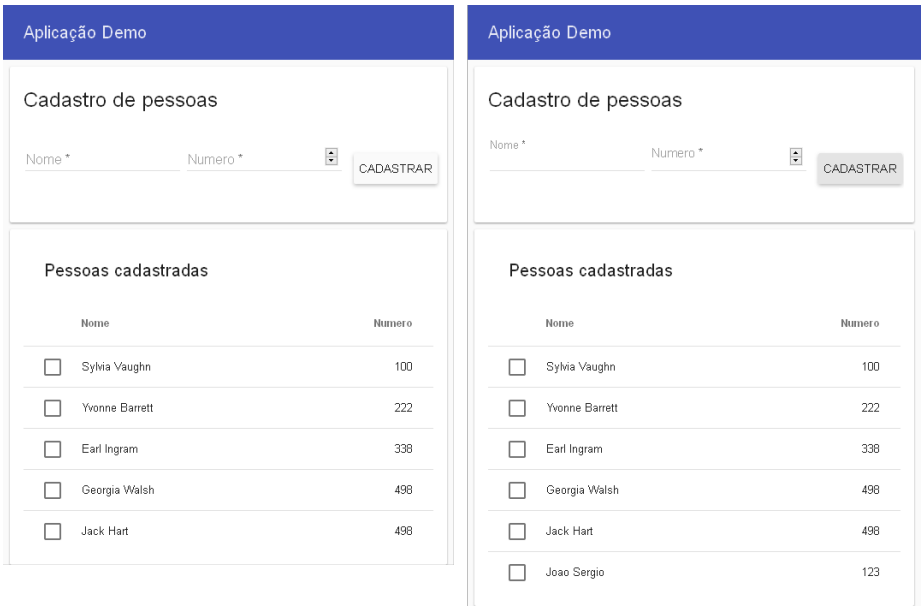

Figura 6: Retrato da execução do teste de cadastro válido.

número fornecido não ser um número. A aplicação deve rejeitar essa entrada. A Figura 7 ilustra a captura da tela antes (lado esquerdo) e depois (lado direito) do teste. É possível observar que o cadastro não foi realizado

Por último, o segundo teste tenta realizar a remoção do primeiro cadastro da aplicação, de nome *Sylvia Vaughn* e número *100*. A aplicação deverá clicar na linha especificada e em seguida no botão *Deletar Selecionado* que será exibido. A figura 8 ilustra a captura da tela antes (lado esquerdo) e depois (lado direito) do teste. É possível observar que o registro foi removido com sucesso.

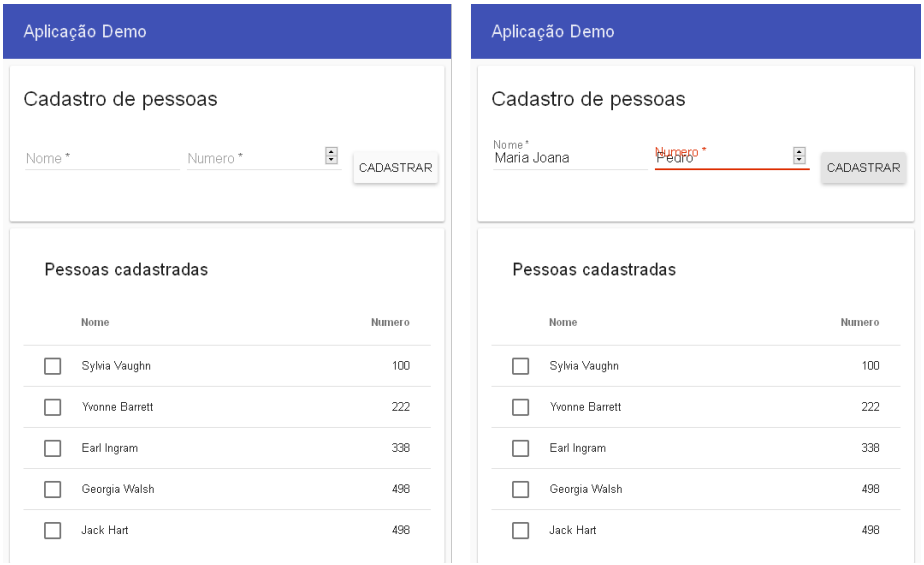

Figura 7: Retrato da execução do teste de cadastro inválido.

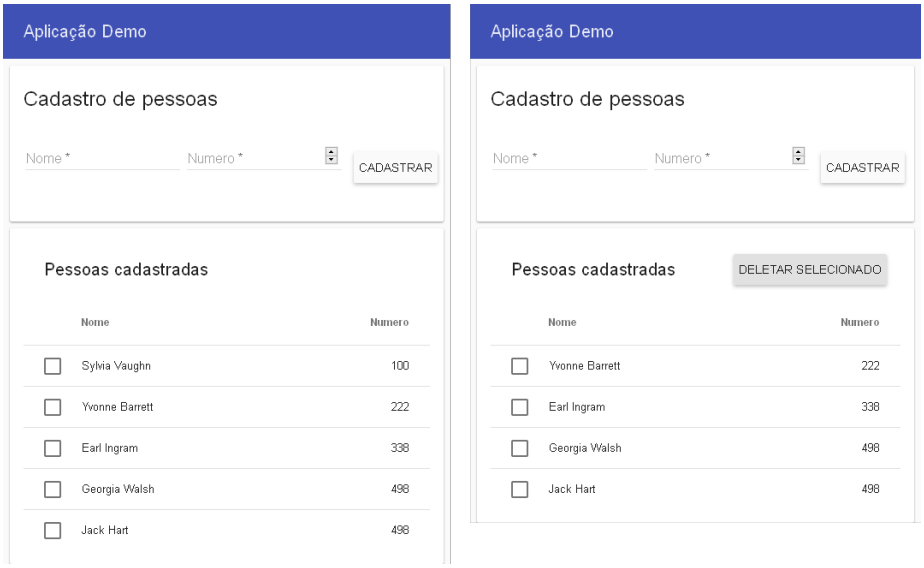

Figura 8: Retrato da execução do teste de remoção de cadastro.

5 Conclusão

Este trabalho apresentou conceitos de aplicações web e teste de software, demonstrando a implementação de uma aplicação com o intuito de automatizar a realização de testes de unidade em aplicações web.

A atividade de teste de software é contornada por obstáculos que dificultam a sua realização. Cada linha de código apresenta um potencial ponto de falha no sistema desenvolvido e novas tecnologias e paradigmas emergem frequentemente, principalmente no ambiente web.

Dado esse cenário, novas ferramentas e abordagens de teste para aplicações web, especialmente as que independem das tecnologias utilizadas na aplicação, como a apresentada neste trabalho, podem se tornar instrumentos de grande valor para o desenvolvimento de software.

Em trabalhos futuros, existem muitos pontos para a expansão da ferramenta, principalmente no que se refere aos métodos de acesso e verificação. Há possibilidade de estender as formas de identificação de elementos na página e definição de fluxos de espera de carregamento de dados assíncronos. Também é possível utilizar esta abordagem para realização de testes de stress e desempenho na aplicação, simulando cargas em paralelo. Além disso, também é possível estender a automatização da ferramenta, incluindo a geração de testes. Eliminando-se assim, mais uma etapa manual do processo de teste de software.

6 Referências

[1] Ajax: A new approach to web applications. `http://adaptivepath.org/ideas/ajax-new-approach-web-applications/`. Acesso em: 2017-02-16.

[2] Beautifulsoup. `https://www.crummy.com/software/BeautifulSoup/`. Acesso em: 2017-06-12.

[3] Everystep scripting tool. `https://www.everystep-automation.com/`. Acesso em: 2017-06-02.

[4] Html+ discussion document - november 8, 1993. `https://www.w3.org/MarkUp/HTMLPlus/htmlplus_1.html`. Acesso em: 2017-02-16.

[5] Ict facts and figures 2016. `http://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx`. Acesso em: 2017-02-15.

[6] Java servlet api specification - version 2.2. `http://web.archive.org/web/20070111213501/http://java.sun.com/products/servlet/2.2/`. Acesso em: 2017-02-16.

[7] Json. `http://www.json.org/`. Acesso em: 2017-06-06.

[8] Mocky. `http://www.mocky.io/`. Acesso em: 2017-06-01.

[9] Phantomjs. `http://phantomjs.org/`. Acesso em: 2017-06-12.

[10] Postman. `https://www.getpostman.com/`. Acesso em: 2017-06-01.

[11] Python 3.6.2 documentation - urllib. `https://docs.python.org/3.6/library/urllib.html`. Acesso em: 2017-06-12.

[12] Sandbox. `https://getsandbox.com/`. Acesso em: 2017-06-01.

[13] Selenium ide. `http://www.seleniumhq.org/projects/ide/`. Acesso em: 2017-06-02.

[14] Selenium webdriver. `http://www.seleniumhq.org/projects/webdriver/`. Acesso em: 2017-06-12.

[15] Xml path language (xpath) 3.1. `https://www.w3.org/TR/xpath-31/`. Acesso em: 2017-06-29.

[16] The xmlhttprequest object. `https://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/`. Acesso em: 2017-02-16.

[17] Alain Abran, Pierre Bourque, Robert Dupuis, e James W. Moore, editors. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2001.

[18] Paul Ammann e Jeff Offutt, editors. *Introduction to Software Testing*. Cambridge University Press, Cambridge, UK, 2008.

[19] Giuseppe A. Di Lucca e Anna Rita Fasolino. Testing web-based applications: The state of the art and future trends. *Inf. Softw. Technol.*, 48(12):1172–1186, dezembro de 2006.

[20] Blaine Donley e Jeff Offutt. Web application testing challenges, 2009.

[21] Mehdi Jazayeri. Some trends in web application development. *2007 Future of Software Engineering*, FOSE 07, páginas 199–213, Washington, DC, USA, 2007. IEEE Computer Society.

[22] The JSON Data Interchange Format. Relatório Técnico Standard ECMA-404 1st Edition / October 2013, ECMA, outubro de 2013.

[23] Haroon S. Oluwatosin. Client-server model. *IOSR Journal of Computer Engineering*, 16(1):67–71, fevereiro de 2014.

[24] Anisha Tandon e Mamta Madan. Challenges in testing of web applications. *International Journal Of Engineering And Computer Science*, 3(5):5980–5984, maio de 2014.

[25] David Vignoni. Client-server model. `https://en.wikipedia.org/wiki/File:Client-server-model.svg`, 2011. Acesso em 2017-07-17.

A Detalhes do automatizador

```
1 Uso: automatizador.py [-h] -i INPUT [INPUT ...] [-p
      ↪ PHANTOMJS] [-o [OUTPUT]] [-d]
2
3 Automatizador de testes de unidade para aplicações web.
4
5 argumentos obrigatórios:
6   -i INPUT [INPUT ...], --input INPUT [INPUT ...]
```

```
7       Documentos de execucao JSON dos casos de teste.
8
9   argumentos opcionais:
10    -h, --help
11        Exibe a mensagem de ajuda e encerra.
12
13    -p PHANTOMJS, --phantomjs PHANTOMJS
14        Caminho com o executavel do PhantomJS. O padrão é ./
              ↪ phantomjs/phantomjs
15
16    -o [OUTPUT], --output [OUTPUT]
17        Armazena o resultado da execução no arquivo
              ↪ especificado.
18
19    -d, --debug
20        Exibe mensagens de controle e execução em stdout, e
              ↪ disponibiliza screenshots da aplicação testada
              ↪  em ./logs
```

Lista 11: Parâmetros de execução da aplicação.