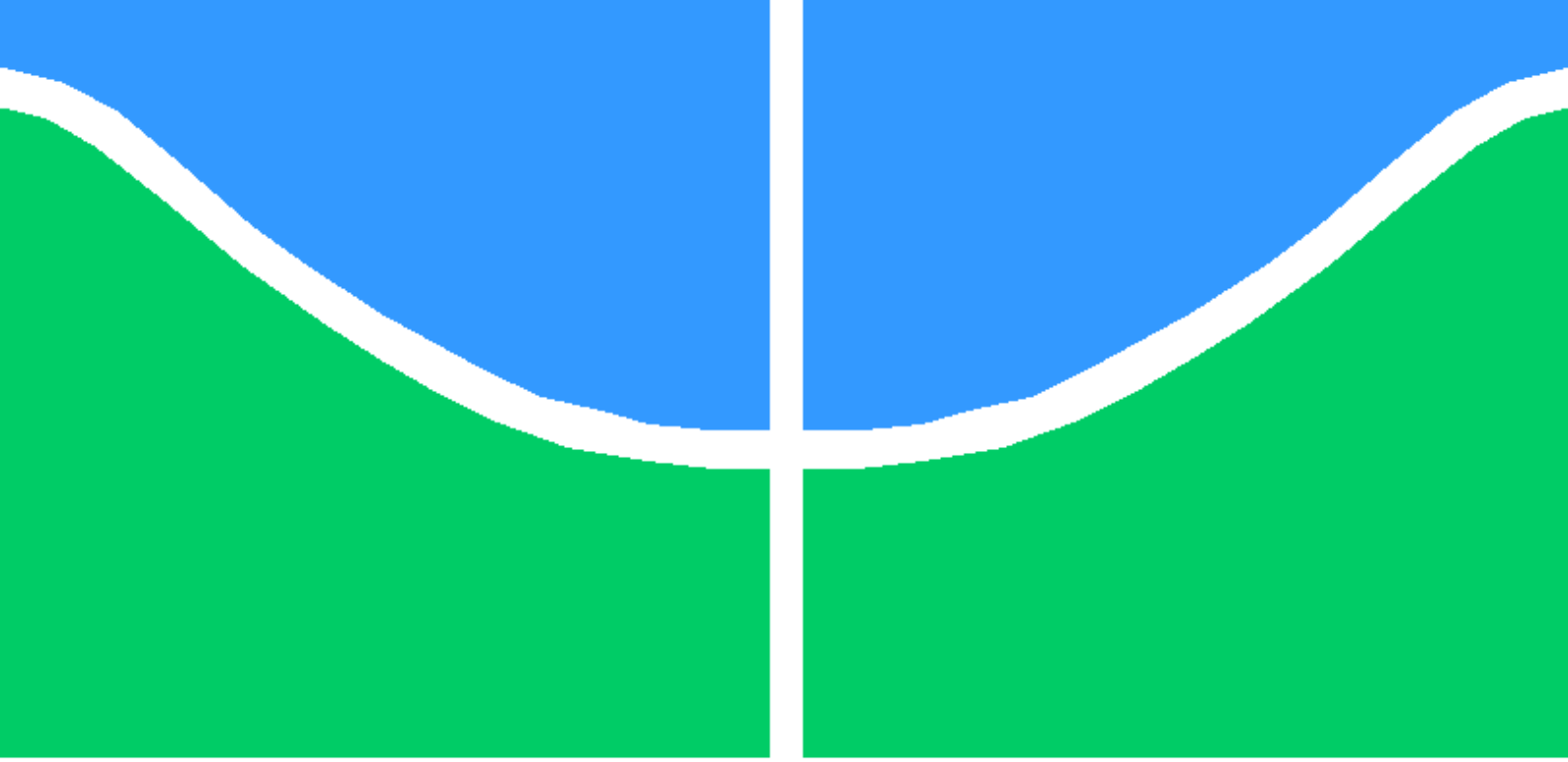

USO DE APRENDIZADO DE MÁQUINA PARA A AUTOMAÇÃO DE TESTES DE SISTEMAS *WEB*

FRANCISCO VITOR LOPES DA FROTA

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

USO DE APRENDIZADO DE MÁQUINA PARA A AUTOMAÇÃO DE TESTES DE SISTEMAS *WEB*

FRANCISCO VITOR LOPES DA FROTA

Orientador: DR. RAFAEL TIMÓTEO DE SOUSA JÚNIOR, ENE/UNB
Co-Orientador: DR. ROBSON DE OLIVEIRA ALBUQUERQUE, ENE/UNB

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO PPGENE.DM - 686/2017
BRASÍLIA-DF, 20 DE DEZEMBRO DE 2017.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

USO DE APRENDIZADO DE MÁQUINA PARA A AUTOMAÇÃO DE TESTES DE SISTEMAS *WEB*

FRANCISCO VITOR LOPES DA FROTA

DISSERTAÇÃO DE MESTRADO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. Dr. Rafael Timóteo de Sousa Júnior, ENE/UNB
Orientador

Prof. Dr. Flávio Elias Gomes de Deus, ENE/UNB
Examinador interno

Prof. Dr. Clarimar José Coelho, PUC-GO
Examinador externo

BRASÍLIA, 20 DE DEZEMBRO DE 2017.

FICHA CATALOGRÁFICA
FRANCISCO VITOR LOPES DA FROTA
Uso de aprendizado de máquina para a automação de testes de sistemas *web*
2017xv, 140p., 201x297 mm
(ENE/FT/UnB, Mestre, Engenharia Elétrica, 2017)
Dissertação de Mestrado - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

FRANCISCO VITOR LOPES DA FROTA (2017) Uso de aprendizado de máquina para a automação de testes de sistemas *web*. Dissertação de Mestrado em Engenharia Elétrica, Publicação 686/2017, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 140p.

CESSÃO DE DIREITOS

AUTOR: FRANCISCO VITOR LOPES DA FROTA
TÍTULO: Uso de aprendizado de máquina para a automação de testes de sistemas *web*.
GRAU: Mestre ANO: 2017

FRANCISCO VITOR LOPES DA FROTA
Quadra 32, Conjunto 24, Lote 28, Apartamento 203

Agradecimentos

Ao meu amigo Daniel, por me dar a oportunidade de entrar no meio acadêmico.

Ao Prof. Dr. Rafael Timóteo de Sousa Júnior e o primeiro orientador Dr. Robson de Oliveira Albuquerque e , pelo apoio e a paciência, que foram indispensáveis para a conclusão deste e outros trabalhos.

Aos meus colegas de trabalho da SOF e Ministério do planejamento que permitiram que eu crescesse como profissional, sobretudo ao Vinícius pelo suporte e aconselhamento nos algoritmos de inteligência artificial.

Gostaria de agradecer também a professora Maristela e ao professor Clarimar pelas contribuições ao trabalho.

RESUMO

USO DE APRENDIZADO DE MÁQUINA PARA A AUTOMAÇÃO DE TESTES DE SISTEMAS *WEB*

Autor: Francisco Vitor Lopes da Frota
Orientador: Dr. Rafael Timóteo de Souza Junior
Programa de Pós-graduação em Engenharia Elétrica
Brasília, Dezembro de 2017

Este trabalho apresenta uma metodologia para a criação de testes de software automatizados. A automação dos testes de software teve como base a utilização de Aprendizado de Máquina, através de Rede Neural Artificial, para o reconhecimento de padrões em páginas HTML. Para identificação das referências de entrada e saída de dados, é utilizado um Banco de dados baseado em grafos, gerando o mapeamento da aplicação sobre teste (do inglês *Application Under Test* - AUT). Para a automatização da geração dos dados de entrada para os testes é utilizada a metodologia de Algoritmo Genético. Os resultados obtidos demonstram, que a partir da metodologia proposta, é possível simplificar a realização dos casos de teste, através de uma linguagem de alto nível. Observou-se também a possibilidade de viabilizar um ambiente de alta performance para realização de testes automatizados de software e interpretação dos resultados através de Processamento Natural de Linguagem (NLP). Como conclusão é possível afirmar que a metodologia proposta, quando comparada a algumas soluções disponíveis no mercado de teste de software, pode atender aos mesmos requisitos e também suprir algumas possibilidades de testes que agregam o processo como um todo.

Palavras chaves: Automação de testes, Redes neurais artificiais, Qualidade de *software*, Algoritmos genéticos, Banco de dados em grafo.

ABSTRACT

USE OF MACHINE LEARNING FOR AUTOMATION O WEB SYSTEMS

Author: Francisco Vitor Lopes da Frota
Supervisor: Dr. Rafael Timóteo de Souza Junior
Post-Graduation Program on Eletrical Engineering
Brasília, December 2017

This dissertation proposes a methodology for automating the process of creating test cases using Machine Learning. This procedure is performed through Artificial Neural Networks, used to identify patterns in HTML pages (Hyper Text Markup Language); identification of Genetic Algorithms, used for the creation of test values; Graphs database are used to perform the mapping; and, finally, general understanding of the application to be tested.

With the bibliographic review carried out in this work, we identified some problems related to the implementation and automation of tests, such as the difficulties of writing and keeping the test cases in operation, and the high costs related to the creation of scripts and maintenance required to ensure the continued operation of test cases. Finally, experiments were carried out to verify the simplification of the language used and the speed of execution of the test cases, as well as the possibility of performing the automation of a good part of the test case creation process.This paper proposes a methodology for automating the process of creating test cases using Machine Learning. This procedure is performed through Artificial Neural Networks techniques, used to identify patterns in HTML (Hyper Text Markup Language) pages ; identification of Genetic Algorithms, used for the creation of test values; Database in graphs, used to perform the mapping; and, finally, general understanding of the application to be tested.

Keywords: Test automation, Artificial Neural Networks, Software Quality, Genetic Algorithms, Graph databases.

SUMÁRIO

Siglas e abreviações

AG Algoritmos Genéticos

AM Aprendizado de Máquina

AUT *Aplication Under Test*

BDD *Behaviour Driven Development*

CRUD *Create, Read, Update, Delete*

CSS *Cascading Style Sheets*

DM *Data Mining*

DOM *Document Object Model*

EUD *End User Programming*

FMS *Finite State Machine*

FIT *Framework for Integrated Test*

FP *False Positive*

GUI *Graphical User Interface*

HTML *HyperText Markup Language*

IA Inteligência Artificial

IR *Information Retrieval*

PLN Processamento de Linguagem Natural

RegEx *Regular Expressions*

RFT Rational Functional Tester

RNA Redes Neurais Artificiais

ROC *Receiver Operating Characteristic*

SIATES Sistema Inteligente de Automação de Testes

SIOP Sistema Integrado de Planejamento e Orçamento

SUT *System Under Test*

SBST Search Based Software Testing

TDD *Test Driven Development*

TD *Table Data*

TH *Table Header*

TP *True Positive*

TR *Table Row*

VRP *Vehicle Routing Problems*

XML *Extensible Markup Language*

XPath *XML Path Language*

XML *Extensible Markup Language*

W3C *World Wide Web Consortium*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS FONTE

Capítulo 1

Introdução

Um teste de *software* pode ser definido como um processo que visa assegurar que uma funcionalidade de um sistema faça o que deve ser feito e nada mais, ou seja, é realizado por meio da execução de um programa, comparando o seu comportamento com o que foi especificado [Myers and Miller 1988].

O processo de automação de testes envolve atividades de criação, execução e análise de resultados. Existem fatores que podem dificultar o processo de automação de teste de software *web* como a velocidade de resposta de carregamento da página *web*, objetos, presentes ou ocultos em uma página *web*, entre outros [Memon et al. 2001].

Existem soluções no contexto de automação de testes que buscam facilitar o processo de automação de testes, no entanto essas ferramentas [Little et al. 2007], [Mahmud and Lau 2010] agem em pontos específicos, enquanto o SIATES unifica essas soluções e faz uso de técnicas de aprendizado de máquina para

A solução proposta neste trabalho objetiva eliminar parte da complexidade associada à criação de testes automatizados. Para tanto, foi utilizado um método de Aprendizado de Máquina (AM) através de tecnologias de Redes Neurais Artificiais (RNA) [Bishop 1995], e também uma técnica relacionada a Inteligência Artificial (IA) denominada como Algoritmos Genéticos (AG) [Coley 1999]. Esse processo utiliza RNA para a identificação automática de objetos de teste, AG para a criação de dados de entrada e conceitos de bancos de dados em grafos para o mapeamento do sistema a ser testado, que pode ser definido como *Aplication Under Test* (AUT) [Parveen and Tilley 2010], que pode ser encontrado também com os termos SUT (*System Under Test*) [Dustin 2002] ou *Software Under Test* [Srivastava and Kim 2009].

Neste sentido, a solução proposta visa unificar as técnicas para automatização de testes de *software* incluindo a criação, execução e análise automatizada, denominado como Sistema Inteligente de Automação de Testes (SIATES).

1.1 Motivação

No decorrer da execução de uma TED (Termo de Descentralização - metodologia de contratação intergovernamental) firmada entre a UnB (Universidade de Brasília) e SOF (Secretaria de Orçamento Federal), na qual o autor participou como pesquisador na equipe de desenvolvimento de *software*. Foi identificado que os casos de testes automatizados eram, inicialmente, escritos sem apoio da documentação do sistema, ou seja, sem o suporte oficial, de modo que os casos de testes automatizados eram escritos com base na aplicação e nas regras de negócio obtidas diretamente do engenheiro de testes e dos desenvolvedores.

Uma pesquisa realizada com a utilização do software Adobe Reader[1] indicou que, entre duas versões consecutivas de um mesmo sistema, 74% dos casos de teste automatizados para interface deixam de funcionar [Memon and Soffa 2003]. Neste mesmo estudo demonstrou-se que uma única mudança em uma página *HyperText Markup Language* (HTML) resulta em erro em 30% a 70% dos testes.

Apesar de existirem trabalhos disponíveis para a geração de testes automáticos de sistema [Anand et al. 2013] e [Kifetew et al. 2013], estas não são compatíveis com testes orientados à interface, sobretudo no que se refere à utilização da integração com AM.

1.2 Objetivos

Este trabalho se propõe a utilizar técnicas de IA e na criação de testes automatizados, a fim de simplificar o processo de automação de testes e reduzir custos no ciclo de desenvolvimento.

1.2.1 Objetivo Geral

Este trabalho de mestrado tem como objetivo a proposta de um sistema baseado em um modelo de Aprendizado de Máquina, Algoritmos Genéticos e Bancos de dados em grafos para automatização de testes de *software Web*, que possibilite a simplificação dos processos de automação de testes podendo melhorar o ciclo de desenvolvimento de um *software*.

1.2.2 Objetivos Específicos

Para atender ao objetivo geral deste trabalho, foram definidos os seguintes objetivos específicos:

- Propor um sistema, denominado SIATES para a criação automática de casos de testes, a fim de diminuir o tempo necessário para testar um sistema e aumentar a qualidade

[1] https://get.adobe.com/br/reader/

final da AUT, de modo que tais casos de testes automatizados possam ser facilmente entendidos e modificados por testadores, sem a necessidade de recorrer a ferramentas complexas e ambientes de desenvolvimento (proposta detalhada no Capítulo 2). A criação de casos de testes estão restritos a funcionalidades de pesquisa e cadastro.

- Aplicar técnicas de RNA, AG, Processamento de Linguagem Natural (PLN) e Bancos de dados em grafo na identificação de objetos de teste, na criação de dados de entrada de teste, na identificação de mensagens de sistema e no mapeamento da aplicação, (resultados detalhados na Capítulo 3).

- Comparar os resultados de casos de testes automatizados gerados pela aplicação e com o comportamento de um testador manual ao executar os testes entre a ferramenta de automação de teste e os engenheiros de testes (procedimento detalhado na Capítulo 5).

1.3 Metodologia de Desenvolvimento

O método aplicado foi o da pesquisa qualitativa e quantitativa, de caráter empírico a partir da pesquisa de tecnologias similares e por meio de experimentos, buscando resultados que possam ser quantificados [Da Silva and Menezes 2005].

O principal objetivo dessa pesquisa foi identificar e comparar, através dos experimentos e da comparação com soluções existentes, os métodos mais eficientes a serem aplicados no campo de qualidade de *software*. Nos experimentos realizados, a base de dados de teste foi composta por *templates* de códigos-fonte reais bem como de um protótipo de aplicação.

Tendo em vista a diversidade dos sistemas *web* o SIATES possui limitações em relação a sua aplicação sendo passível de utilização somente em cenários de cadastro e pesquisa de dados. Com base nos resultados obtidos, é possível identificar o funcionamento esperado nos seguintes cenários:

- Localização de objetos de teste: o processo de identificar um objeto de teste dentro de uma página *web* garantindo a sua unicidade.

- Classificação das mensagens de sucesso ou erro que são exibidas pela AUT após a execução de uma tarefa, também denominadas como mensagens de *status*;

- Geração de dados de entrada: dados que são usados especificamente para teste de *software*, que são a entrada de um formulário em um sistema, para validação de um resultado esperado;

- Mapeamento de sistema: o processo de identificar as entradas (entrada de teste) e saídas de dados (resultados de teste) e a ligação entre eles;

No projeto inicial, os casos de teste eram escritos independentemente da documentação do sistema, ou seja, sem o suporte oficial, de modo que os casos de testes automatizados eram escritos com base no sistema e nas regras de negócio obtidas diretamente do engenheiro de testes e dos desenvolvedores. O objetivo, portanto, era fornecer meios para que os testadores pudessem executar os testes, detectar falhas e reportar esses resultados à equipe de desenvolvimento.

Considerando um cenário onde uma sistema *web* já exista e que hajam dados reais armazenados através do seu uso, os experimentos propostos através do sistema SIATES apresentado no Capítulo 5, possibilitam a criação automatizada de casos de testes utilizando tecnologias de Aprendizado de Máquina em conjunto com Algoritmos Genéticos e Processamento de Linguagem Natural.

1.4 Organização do trabalho

Essa dissertação é organizada da seguinte forma: o Capítulo 2 apresenta os conceitos necessários para o entendimento da solução apresentada, com foco em AM, RNA, automação de testes e AG. O Capítulo 3 detalha os problemas encontrados durante a implementação da automação de testes no ciclo de desenvolvimento e as possíveis soluções para tais problemas. O Capítulo 4 detalha a arquitetura usada para o desenvolvimento da solução proposta. O Capítulo 5 descreve os experimentos realizados e os seus resultados. Por fim, o Capítulo 6 apresenta as conclusões do trabalho e os objetivos alcançados.

Capítulo 2

Referencial teórico

Este Capítulo apresenta os conceitos teóricos necessários para a compreensão da solução apresentada nesta dissertação . Aqui, explicam-se conceitos de qualidade e testes de *software* que são usados como base e como justificativa para a criação do SIATES. Explicam-se, também, as técnicas e os conceitos das ferramentas usadas para a criação do SIATES, como automação de testes, Bancos de dados em grafo, RNA e PLN

2.1 Qualidade de *software*

Um programa de *software* corresponde a um grupo de processos computacionais que realizam um trabalho intelectual. Um programa que funciona de maneira correta executa seus processos de forma precisa e para que o programa funcione de forma correta, é necessário certificar sua qualidade [Abelson et al. 1996].

Qualidade de *software* reflete o quão bem um programa está em conformidade com uma série de requisitos e especificações funcionais, a qualidade de um programa pode ser assegurada através da realização de teste de *software* [Pressman 2005], que pode ser feito de forma manual ou automatizada.

2.1.1 Testes e controle de qualidade

Teste de *software* é o processo de executar um programa com o intuito de validar sua funcionalidade. Esse processo é realizado por meio da execução do programa, do registro e da comparação do seu comportamento com o que está especificado [Myers and Miller 1988], a validação da qualidade de um programa por meio de um teste pode ser feita de forma manual ou automatizada.

No teste manual uma pessoa assume o papel de testador para avaliar o comportamento do programa analisando as circunstancias em que o programa não se comporta como deve [Nidhra and Dondeti 2012].

Já em testes automatizados, o *software* executa o processo para validar as funcionalidades de um sistema e sua qualidade. Um teste automatizado não requer intervenção humana para sua execução, no entanto é necessário que o caso de teste, que é o conjunto de condições e especificações de conformidade para o sistema, seja criado previamente. A automação de testes é capaz de executar os testes, realizar comparações e exibir os resultados, de modo que esses testes podem ser executados sempre que solicitado ou automaticamente. O *software* de automação age diretamente sobre a AUT, onde interage com os objetos de teste.

Objeto de teste é um componente individual do sistema a ser testado. No caso de teste em aplicações *web*, um objeto de teste pode ser um botão, um campo de texto, uma caixa de seleção ou um *link*. Automação dos testes apresenta diversas vantagens em comparação ao teste manual, conforme apresentado a seguir.

- Confiabilidade: os testes são executados com base em um caso de teste, portanto executam precisamente as mesmas operações todas as vezes que são requisitados. Em contrapartida, testadores manuais podem cometer erros devido ao cansaço ou à falta de atenção [Rafi et al. 2012].
- Velocidade : testes automatizados são geralmente mais rápidos que testadores manuais [Rafi et al. 2012].
- Repetibilidade: os testes, após serem criados, podem ser executados repetidamente sempre que requisitados ou, ainda, automaticamente [Rafi et al. 2012].

2.1.2 Automação de testes para aplicações *web*

Aplicações *web* são compreendidas aqui como programas de *software* que são usados por meio de um navegador, o qual exibe os objetos de teste que são interpretados via HTML. HTML, que é uma linguagem de marcação para publicação de conteúdo na Internet, executa sua função por meio de um conjunto de marcações inserido em um arquivo disponibilizado *online*, por meio de padronizações da *World Wide Web Consortium* (W3C) [1], uma unidade reguladora envolvida no desenvolvimento de padrões para internet.

As páginas HTML são organizadas hierarquicamente por meio de uma estrutura lógica denominada *Document Object Model* (DOM). DOM é uma especificação para o documento que define como ele pode ser acessado e manipulado. Com a utilização desse padrão, é possível adicionar, modificar e deletar objetos de teste e conteúdos. No caso de ferramentas de automação, o DOM é usado para localizar os objetos de teste na página HTML.

Para que uma ferramenta de automação possa obter o valor de um objeto HTML, ela pode navegar na hierarquia DOM por meio de uma linguagem, denominada Sintaxe para navegação em documentos XML (XPath), que permite a seleção de nós de uma hierarquia.

[1] https://www.w3.org/TR/1999/REC-html401-19991224/intro/intro.html

XPath é um padrão proposto pela W3C para a localização de objetos de teste em uma estrutura de árvore [Gottlob et al. 2005]. Originalmente usado para obter e manipular dados em documentos estruturados hierarquicamente em XML, este padrão foi adaptado para ser usado em páginas HTML por ferramentas de automação.

Teste automatizado para aplicações *web* é uma modalidade de teste de caixa preta que permite a execução de teste sem intervenção humana. O alvo dos testes é a interface gráfica da aplicação. Durante a execução dos testes, cliques de *mouse* e comandos de teclados são executadas no navegador para o preenchimento de um campo ou o clique em um botão.

2.2 Aprendizado de Máquina e reconhecimento de padrões

Em sua definição básica, com Aprendizado de Máquina um programa de computador aprende através da experiencia (E), para realizar alguma tarefa(T) com uma medida de performance (P), que melhoram ao passar do tempo com experiencia(E). Uma das utilidades do uso de aprendizado de máquina envolve a busca de uma solução em um espaço grande de soluções para selecionar a melhor para um determinado problema [Mitchell 1998].

Aprendizado de Maquina automatiza o desenvolvimento de modelos analíticos usando algoritmos que podem aprender interativamente a partir de dados. Este método permite que computadores identifiquem padrões sem serem explicitamente programados, melhorando seu próprio desempenho a partir das experiências às quais são submetidos, assim definindo sua capacidade de aprender, usando o conceito de aprendizado de máquina [Michalski et al. 2013].

O aprendizado supervisionado contém um recurso humano ou automático que faz a supervisão do processo. A partir desse processo de supervisão uma saída conhecida como classe é dada ao conjunto de dados de entrada, de forma a adaptar os neurônios a esses estímulos criando um classificador a partir da RNA [Widrow and Hoff 1960]. Independentemente do processo a ser realizado, a implementação de uma RNA possui alguns passos em comum, conforme apresentado a seguir:

1. Desenvolver modelo de treino: inicialmente, é necessário definir como a experiência do sistema será adquirida, o que terá impacto direto na eficácia do algoritmo. É importante definir como o sistema interpreta a resposta do problema e como o algoritmo irá usá-la para melhorar. O *software* pode aprender por meio de diversas fontes, como, por exemplo, um especialista ou exemplos previamente executados.

2. Definição do alvo de função: define o tipo de conhecimento que será aprendido pelo algoritmo. Dado um contexto, o algoritmo deverá ser capaz de criar soluções válidas bem como escolher a melhor solução entre as soluções de um grupo.

3. Criar a função de aproximação : com a utilização dos exemplos disponíveis, é neces-

sário definir a aproximação de uma solução ideal. Na maioria dos casos, é possível determinar se uma solução é adequada, de modo que um funcionamento preciso está relacionado à obtenção de diferentes graus de sucesso.

2.2.1 Terminologias de Redes Neurais Artificiais

Nesta Subseção, são explanados os conceitos necessários para o entendimento das terminologias relacionadas a RNA.

- Generalização: essa terminologia se refere à capacidade de generalizar os dados, de forma que a rede possa lidar com dados que nunca foram vistos antes, abstraindo propriedades de dados existentes na rede que contém o mesmo padrão.
- Mapeamento de entrada e saída: essa terminologia diz respeito à técnica de treinamento supervisionado que objetiva relacionar os neurônios de entrada e de saída utilizando uma amostra de dados do mundo real. No decorrer do treinamento, os pesos sinápticos são adaptados de forma a minimizar a diferença entre a entrada e a saída da rede [Haykin and Network 2004].
- Adaptabilidade: essa terminologia refere-se à capacidade da RNA de lidar com variações mínimas no seu modelo, mantendo a sua capacidade original de classificação, de forma que a RNA possa mudar os seus pesos sinápticos em tempo real e, por conseguinte, possa se adaptar ao contexto do problema.

O neurônio é a unidade base do modelo de processamento de uma RNA. Uma combinação linear é usada para somar as entradas do sinal de acordo com os pesos das respectivas sinapses e uma função de ativação é responsável por definir a limitação do sinal do neurônio, limitando o alcance do seu sinal dentro da rede.

Existem diversas arquiteturas para serem aplicadas em redes neurais, essas arquiteturas partilham uma estrutura básica consistindo de neurônios de entrada, saída possivelmente de neurônios ocultos, dentre as arquiteturas disponíveis existem *Feedforward* [Bebis and Georgiopoulos 1994], *Regulatory feedback* [Xu et al. 2007], *Recurrent neural network* [Medsker and Jain 2001], Modulares [Kimoto et al. 1990], entre outros. O escopo desse trabalho faz o uso somente de RNA com Perceptron multicamada, detalhado a seguir.

2.2.2 Perceptron de Múltiplas Camadas

Perceptron Multicamada é um dos tipos de RNA que consistem em múltiplas camadas de neurônios que interagem entre sí através de conexões baseadas em peso, além das camadas de entrada e saída de dados. Uma rede pode conter camadas ocultas ou intermediárias de modo que não haja interconexões com os neurônios da mesma camada enquanto todos os neurônios

de uma camada estão conectados com as camadas adjacentes, os pesos dos neurônios são usados para medir o grau de correlação entre os neurônios em que eles se conectam.

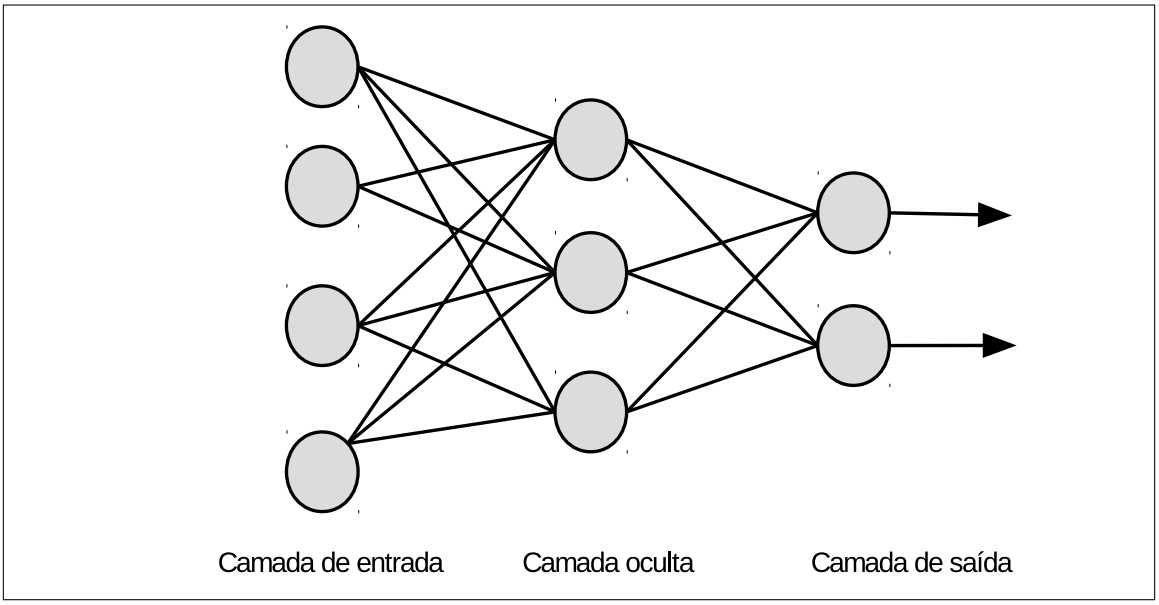

Figura 2.1: Estrutura organizacional de uma Perceptron de Múltiplas Camadas.

A arquitetura de uma rede de Perceptron Múltiplas Camadas, é composta por uma camada de entrada que contêm neurônios sensoriais, responsáveis por propagar os sinais de entrada para uma ou mais camadas ocultas. A camada oculta realiza a combinação linear dos sinais das camadas ocultas e de entrada de acordo com a função de ativação usada [Processing 1986].

A Figura 2.1 mostra uma rede com uma camada oculta, possuindo quatro valores de entrada, três neurônios em sua camada oculta e duas saídas possíveis, os neurônios estão altamente conectados, onde cada um deles está relacionado a sua camada subsequente, característica comum nesse modelo de RNA.

2.3 Bancos de dados em grafo

Um grafo é um conjunto de vértices e arestas representados como nós e relacionamentos. Essa estrutura pode ser usada em uma base de dados, de modo que tipos de cenários que se beneficiem da estrutura de grafo. De acordo com Robinson [Robinson 2015], dados de redes sociais são facilmente representados com grafos, sendo os usuários representados como nós, e as interações representadas como relacionamentos. A Figura 2.2 exemplifica a interação entre os diferentes registros de um banco de dados em grafo.

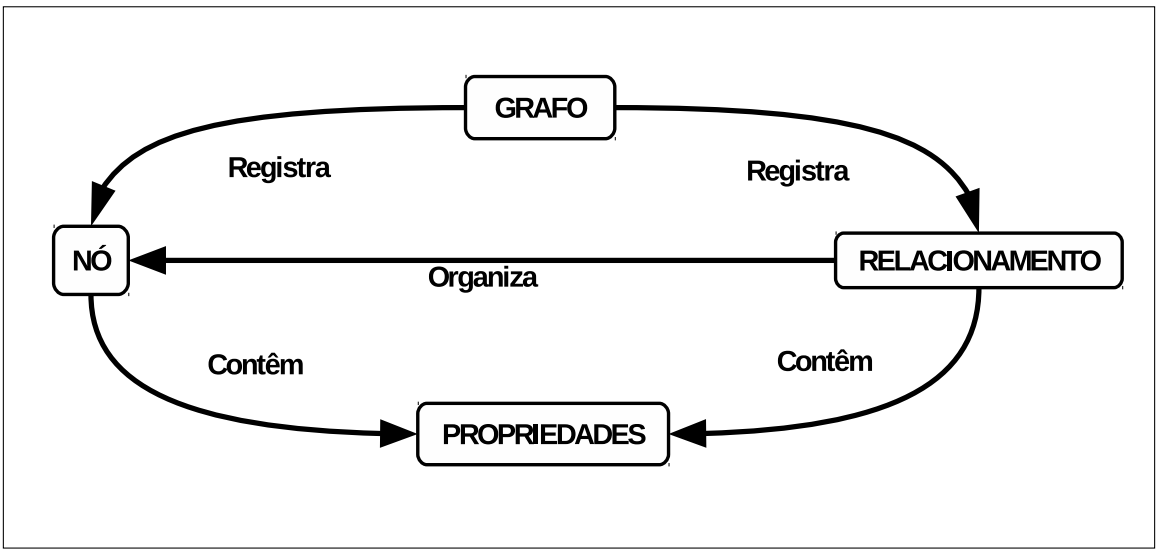

Figura 2.2: Estrutura organizacional de um banco de dados em grafo [Neo4J 2016].

A Figura 2.2 representa os diferentes relacionamentos entre os registros de uma base de dados em grafo. Dentro de uma base de dados são armazenados nós e relacionamentos, sendo os nós organizados por meio dos relacionamentos, o que possibilita armazenar valores nas propriedades dos nós e dos relacionamentos.

O sistema gerenciador de banco de dados em grafos permite fazer as operações básicas de CRUD, que são aplicados a um modelo de dados baseado em grafo. Entre os focos do modelo de grafos incluem-se os relacionamentos, cujas criações, diferentemente de outros bancos de dados em que se faz necessário definir chaves estrangeiras, são embutidas no processo de modelagem, resultando em modelos mais simples para esse tipo de uso [Robinson 2015]. Devido a sua ênfase em grafos [Wood 2012] e [Barceló Baeza 2013], sistemas gerenciadoras de banco de dados em grafos são otimizadas para escanear e processar grandes massas de dados altamente conectados por relacionamentos. A tecnologia permite que o modelo possa ser evoluído facilmente, de modo que adicionar uma propriedade ou expandir uma estrutura não resultará em erros nas *queries* e funcionalidades já existentes [Van Bruggen 2014].

2.4 Algoritmos genéticos

Algoritmos Genéticos (AG) são algoritmos de otimização baseados nos modelos de seleção natural e genética, ambos inspirados nas teorias de Darwin [Darwin 1872]. Esses algoritmos podem ser aplicados a uma grande quantidade de problemas, inclusive problemas práticos do dia a dia [Coley 1999]. Neste trabalho, AG são usados para criar cadeias de caracteres, as quais serão utilizadas nos testes para o preenchimento de campos de texto. Essas cadeias de caracteres são referidas neste trabalho como dados de entrada de teste. Aplicações bem-sucedidas de AG incluem:

1. Processamento de imagem [Sonka et al. 2014], [Paulinas and Ušinskas 2015]

2. *Vehicle Routing Problems* (VRP) [Vidal et al. 2013],

3. Robótica [Ayala and dos Santos Coelho 2012], [Tuncer and Yildirim 2012]

4. Reconhecimento Facial [Bhatt et al. 2013], [Al-Arashi et al. 2014]

Holland [Booker et al. 1989] definiu quatro elementos básicos para AG. Apesar de não existir uma estrutura rigorosa para a definição de um algoritmo genético, muitos dos AG contêm os seguintes elementos em comum [Mitchell, 1998]:

- Um grupo de possíveis soluções para o problema, denominado população;
- Um teste que define a qualidade da solução para cada uma das soluções da população;
- Um método para juntar as partes das melhores soluções para formular soluções ainda melhores;
- Um fator de mutação para evitar perda permanente entre as soluções.

2.4.1 Terminologia algorítmica

O termo cromossomo é usado dentro de um AG para representar uma solução candidata e, geralmente, é representado como uma sequência binária. Um cromossomo é composto por múltiplos genes, ou seja, se o cromossomo for uma possível solução em um formato binário, o gene vai ser um '0' ou '1'. As soluções são obtidas de uma lista possivelmente infinita de soluções possíveis para a resolução de um problema; esta lista de soluções é denominada espaço de busca.

Fitness é a qualidade potencial do cromossomo para a resolução de um problema [Deb et al. 2002]. Tal qualidade é representada de uma forma numérica, sendo resultado de uma função de *fitness*, que é uma função que retorna à qualidade de um cromossomo.

O *fitness* máximo a ser alcançado denomina-se máximo global, ao passo que o pico menor denomina-se máximo local. Para algumas aplicações, obter o máximo global não é necessário para resolver um problema, desde que a solução tenha uma quantidade mínima de *fitness* para aquele problema específico [Deb et al. 2002]. As operações básicas de um AG incluem seleção, *crossover*, mutação, RNA e reconhecimento de padrões, conforme explicitado a seguir.

2.4.2 Seleção

Seleção é a fase em que os cromossomos são selecionados dentro de uma população, simulando seleção natural em tipos biológicos. A primeira população é geralmente composta por cromossomos gerados aleatoriamente [Bodenhofer 2003]. Os indivíduos são ordenados

com base em seu *fitness*; os genes com menor *fitness* têm uma maior chance de serem descartados, ao passo que os genes com maior *fitness* têm uma maior chance de serem usados na próxima geração da população.

2.4.3 *CrossOver*

Crossover é um processo baseado na reprodução animal, em que os materiais genéticos de dois elementos são aleatoriamente divididos e misturados para gerar um novo indivíduo. Assim, elementos que tenham alto *fitness* podem gerar filhos com uma maior qualidade ou, ainda, melhores do que os pais [Bodenhofer 2003].

2.4.4 Mutação

Mutação é o processo que garante a variação e a inovação do AG [Booker et al. 1989], em que um ou mais genes são alterados para manter a diversidade genética.

2.5 Processamento de linguagem natural e classificação de texto

Processamento de Linguagem Natural (PLN) é uma área de pesquisa e aplicação que explora como os computadores podem ser usados para compreender e manipular linguagens naturais para executar tarefas [Chowdhury 2003]. Neste trabalho, processamento de linguagem natural será aplicado na classificação de texto.

A classificação de texto é a tarefa de atribuir a cada um dos textos fornecidos uma categoria de um conjunto pertencente a categorias predefinidas. A classificação binária é o caso mais simples de classificação, em que cada texto fornecido é atribuído a uma das duas categorias predefinidas [Zhu et al. 2005].

No processo de classificação de texto, é estimada a distribuição condicional dos rótulos de cada um dos documentos. Um rótulo ou classe é um nome dado a uma das categorias predefinidas em que um texto poderá ser classificado. Um texto, ou documento, é representado pelo texto que contém o posicionamento e a contagem de palavras. A premissa básica da entropia máxima é que são preferíveis os modelos mais uniformes capazes de satisfazer quaisquer restrições específicas.

Por exemplo, considerando que, em média, 40% dos documentos com a palavra 'professor' estão na classe de documentos rotulada 'faculdade', intuitivamente é possível afirmar que, se um documento contém a palavra professor, ele tem 40% de chance de ser um documento com a classe de 'faculdade'; porém, se o documento não contiver a palavra 'professor', deve-se assumir que existe 60% desse texto pertencente a outra classe. Esse modelo é

conhecido como modelo de entropia máxima. Nesse exemplo, o cálculo é muito fácil, mas a complexidade aumenta em situações com um maior número de classes. Em geral, a Entropia Máxima pode ser usada para estimar qualquer distribuição de probabilidade, mas, neste trabalho, será usada para realizar a classificação entre duas classes.

Um aspecto característico da estimação estatística é que ela não faz suposições independentes: por exemplo, na frase 'Buenos Aires', essas duas palavras estão quase sempre posicionadas juntas, de modo que a evidência dessa frase será duplicada [McCallum et al. 1998].

2.6 Ferramentas utilizadas

Nessa sessão são detalhadas as ferramentas usadas nos experimentos realizados e na construção do SIATES, assim como os pontos em que são usados.

2.6.1 Weka

Weka [2] é uma ferramenta composta por uma coleção de algoritmos de *machine learning*. A ferramenta foi desenvolvida de forma que o usuário tenha possibilidade de testar múltiplos algoritmos e *datasets*, de forma flexível, por meio de sua biblioteca Java ou interface gráfica.

O Weka é usado para executar os algoritmos de RNA usados no experimento do Capítulo 5, para realizar a classificação dos objetos de teste, seu uso dentro da arquitetura do SIATES é detalhado no Capítulo 4.

2.6.2 Neo4j

Neo4j[3] é um banco de dados orientado a grafos que pode ser usado em diversos domínios como, por exemplo, análise social, sistemas de recomendação, detecção de fraudes e roteamento.

No SIATES o Neo4j é responsável por mapear o sistema, seu funcionamento é detalhado no Capítulo 4, e um experimento fazendo o seu uso é feito no Capítulo 5.

2.6.2.1 OpenNLP

OpenNpl [4] é um *framework* usado para o Processamento de Linguagem Natural (PNL). A ferramenta suporta muitas das funcionalidades de PLN, como, por exemplo, tokenização e

[2] http://www.cs.waikato.ac.nz/ml/weka/
[3] https://neo4j.com/
[4] http://opennlp.apache.org/

classificação. Classificação de documentos busca agrupar diferentes conteúdos em categorias que contêm as mesmas características. No SIATES o OpenNLP usado para classificar as mensagens que a AUT exibe após o término de uma operação,

2.6.3 Selenium

Selenium é uma ferramenta de automação para *browsers*, cuja finalidade é realizar a automatização de aplicações para testes e para a realização de quaisquer outras tarefas repetitivas. É composta por duas partes principais: o SeleniumIDE, um *plugin* para o navegador que permite a criação rápida de casos de teste automatizados e são suficientes em caso de testes simples, pois oferece uma interface que permite implementar a maioria dos comandos sem o uso de programação; e o Selenium WebDriver, que pode ser implementado em múltiplas linguagens de programação e permite aplicações mais robustas como regressão de testes. No SIATES o Selenium é responsável por obter os dados da AUT e realizar os testes automatizados.

2.6.4 JSF

JavaServer Faces[5] (JSF, ou simplesmente '*Faces*') é um *framework* de desenvolvimento de aplicativos da *web* que contém componentes de interface do usuário como caixas de texto, caixas de listagem, painéis com guias e dados grades, utilizados no âmbito do desenvolvimento *web*. O JSF é parte do Java *web*, também conhecido como *Java Enterprise Edition* (J2EE). JSF é composto por um conjunto padrão de componentes de interface gráfica, como, por exemplo, botões, *hiperlinks*, caixas de seleção e campos de texto, e permite também a criação de componentes customizados. Este *framework* compõe a interface do SIATES juntamente com o Primefaces.

2.6.5 Primefaces

PrimeFaces[6] é uma biblioteca leve de componentes JSF. A biblioteca disponibiliza uma série de componentes extras aos componentes já disponíveis do JSF, porém com a possibilidade de utilização do recurso temas, que é um recurso de renderização parcial de página que permite que os objetos de teste em páginas possam ser atualizados sem que toda a página seja atualizada.

[5] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[6] https://www.primefaces.org

Capítulo 3

Análise do problema e proposta da solução

Este Capítulo descreve os principais problemas encontrados na implementação de testes automatizados. Os problemas mencionados neste Capítulo afetam diretamente a qualidade da AUT.

3.1 Análise dos problemas

Nesta Seção, são apresentados alguns problemas que comumente são identificados durante a implementação da automação de testes no desenvolvimento de uma aplicação.

3.1.1 Alta complexidade devido ao uso de uma linguagem de programação

Linguagens de programação impõem restrições em sua sintaxe quando são interpretadas por um computador, por exemplo, a linguagem de programação Java usa ';' no fim de cada comando e, caso o programador esqueça desse detalhe em qualquer uma das linhas, o programa não irá compilar. Esses pequenos detalhes dificultam a criação dos casos de testes automatizados, tornando-os mais complexos e menos flexíveis.

Parte das ferramentas de automação atuais usam uma linguagem de programação para a criação de seus casos de testes, o que eleva o custo da mão de obra. Sua implementação requer que o testador tenha conhecimentos de qualidade e programação para que ele possa manipular os casos de testes [Douce et al. 2005].

Um caso de teste é composto por ações como cliques e comandos de teclados, e localizadores, um localizador é um recurso usado pela ferramenta de automação para identificar e localizar um objeto de teste entre todos os outros objetos de testes da página. A Figura

3.1 exemplifica alguns localizadores comuns, na linha 1, o localizador faz referência a uma propriedade que pode mudar em uma versão posterior do sistema, fazendo que o caso de teste perca a referência com o objeto de teste tornando o caso de teste inutilizável.

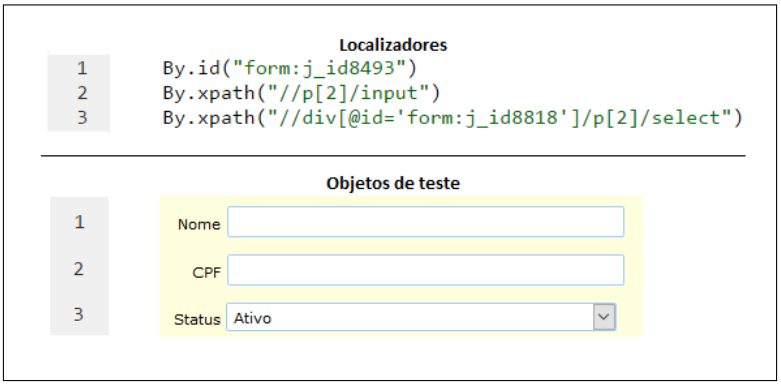

Figura 3.1: Exemplo de localizadores e seus respectivos objetos de teste.

Na Figura 3.1, o localizador 1 está associado ao objeto de teste 1. O objeto de teste 1 é um campo de texto com um texto 'Nome' ao seu lado. No entanto, ao visualizar o localizador da linha 1, não é possível determinar se ele faz referência a um campo chamado 'Nome'. O mesmo ocorre com os objetos de testes subsequentes.

Ainda com relação à Figura 3.1, não fica claro para o testador se o localizador '//p[2]/input' faz referência a um tipo do campo 'CPF', pois dentro do localizador não há referência ao texto 'CPF' ; ao invés disso, a expressão diz somente que o objeto que está sendo procurado é um elemento de entrada de texto, que corresponde ao segundo elemento do formulário. Por utilizar uma linguagem de programação e localizadores pouco representativos

3.1.2 Baixo tempo de vida dos casos de teste

O tempo de vida de um caso de teste é o tempo total em que um caso de teste é capaz de testar uma funcionalidade de software corretamente.

Fatores como casos de teste mal escritos, podem fazer com que um caso de teste possa deixar de funcionar corretamente, encerrando o seu tempo de vida. Em alguns casos, o impacto dos casos de teste mal escritos é grande, de modo que 75% dos casos de teste são modificados em média três vezes antes de serem deletados. Duas das maiores razões para um caso de teste quebrar/falhar são mudanças de posição em tela. Realizar uma simples mudança de posição do elemento no HTML, pode causar a quebra de 30% a 70% dos casos de teste associados àquela interface [Mahmud and Lau 2010].

3.2 Implementação da solução

Previamente ao desenvolvimento do SIATES, foram consideradas outras técnicas de automação de testes. Essas técnicas buscam resolver os problemas encontrados no escopo de testes de *software* e solucionam de forma individual, e não de forma integrada como ocorre no SIATES, alguns dos problemas já apresentados.

3.2.1 Alternativas a uma linguagem de programação

End User Programming (EUD) permite que usuários que não possuam conhecimento técnico em programação de *software* sejam capazes de escrever programas [Haykin and Network 2004]. Essa alternativa já foi usada em conjunto com a automação de testes [Bolin and Miller 2005] e foi implementada também aqui no SIATES.

O SIATES não faz uso direto do caso de teste automatizado do usuário. Em vez disso, a entrada é processada, de modo a mesclar o que foi informado pelo usuário, com configurações predefinidas e informações da própria AUT. O uso de tal linguagem elimina as complexidades relacionadas à utilização de uma linguagem de programação.

3.2.2 Execução facilitada de comandos

Em ferramentas de automação de testes os comandos são executados a partir dos objetos de teste. No SIATES, tais comandos são filtrados e associados com palavras-chave definidas pelo usuário. Essa alternativa evita que o usuário final tenha de lidar diretamente com objetos de teste e as chamadas de método via código-fonte.

A junção dos fatores simplificação das chamadas de comando e simplificação da localização dos objetos de teste resulta na diminuição direta dos custos de automação e na complexidade da criação dos casos de testes.

3.2.3 Busca simplificada de objetos de teste

Diferentemente dos localizadores apresentados anteriormente, o SIATES propõe um novo método para nomear e localizar os objetos de teste. Ao invés de referenciar esses objetos de teste por propriedades internas do HTML, o SIATES busca referenciar os objetos pelas propriedades visíveis ao usuário final, como o texto contido em um botão.

O uso de localizadores de teste simplificados tem efeito direto na redução da complexidade na criação do caso de teste, no que se refere ao objeto de teste em si, aumentando também o tempo de vida do teste, por não fazer uso de localizadores de propriedades e por dispensar a necessidade da página. A Tabela 3.1 exemplifica o padrão de nomeação dos diferentes objetos de teste.

Tabela 3.1: Tipos de elementos de busca.

Tipo de elemento	Descrição
Botões, links e elementos clicáveis	Para esses objetos, o texto contido no elemento de teste é recomendado, pois essa é a propriedade que é exibida ao usuário final.
Elementos identificados por *label* (*checkboxes*, radio-buttons e texto, campos de texto, *selects*, etc.)	São objetos de testes cujos nomes não estão contidos no HTML do próprio objeto de teste, ou seja, ele depende de um texto externo posicionado próximo a esse objeto de teste.

Para os objetos de teste nomeados por uma *label* que é o texto que acompanha o objeto de teste, a associação desses elementos é feita de múltiplas maneiras, como está exemplificado na Figura 3.2. A primeira forma de associação, exemplificada no número 1, da Figura 3.2, mostra dois objetos de teste: o primeiro, do tipo *label*; e o segundo, um campo de entrada de texto.

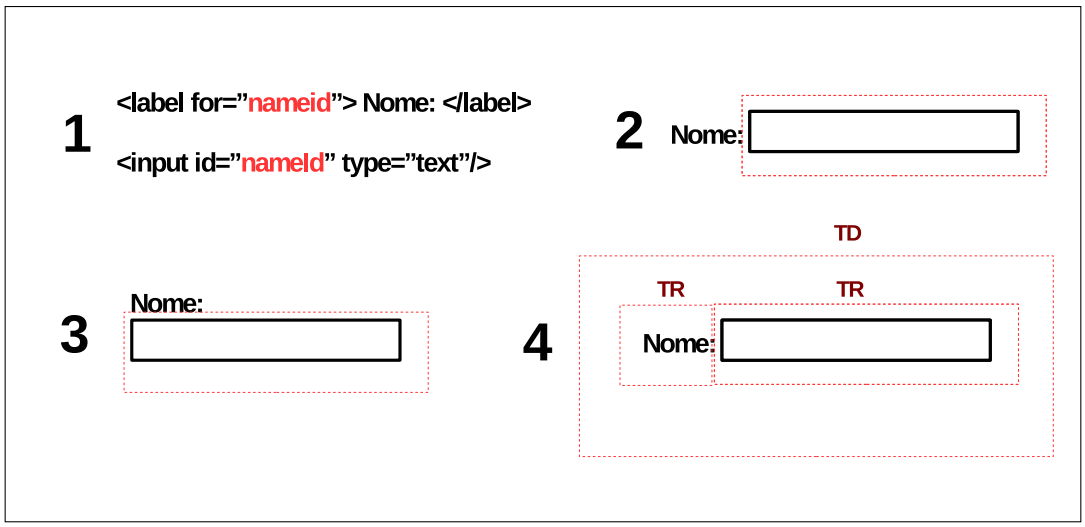

Figura 3.2: Exemplo de localização de objeto externo.

O valor da propriedade *'for' 'nameid'*, marcada em vermelho na primeira linha, é o mesmo valor da propriedade 'id' do campo de texto da segunda linha, de modo que, ao encontrar essa ligação, o SIATES irá associar esses elementos, para que o campo de texto possa ser referenciado pelo valor 'Nome:', contido no objeto de teste de '*label*'.

O exemplo 2, na Figura 3.2, mostra a associação feita por posicionamento na página, alternativa que é usada caso os objetos não contenham as propriedades do exemplo 1. No exemplo 2, o SIATES busca o elemento do tipo *label*, que é o texto 'Nome:', posicionado ao lado do campo de texto, e busca um objeto de teste ao seu lado, campo de busca que é representado pelo retângulo pontilhado vermelho. Caso encontre o objeto de teste próximo a ele, o objeto de teste será identificado com o valor da '*label*', ou seja, 'Nome:'. O mesmo

ocorre no exemplo 3 nas situações em que o objeto de teste está posicionado abaixo da *label*. O exemplo 4, ainda da Figura 3.3, mostra a hierarquia HTML dos objetos, de modo que os retângulos pontilhados vermelhos representam esses elementos HTML. Hierarquicamente, a label e o campo de textos estão mais próximos, o que faz que eles sejam associados pelo SIATES.

3.2.4 Aumento do tempo de vida dos testes

Muitas das limitações associadas ao tempo de vida dos casos de testes estão diretamente relacionadas aos localizadores mal escritos dos objetos de teste. Como já mencionado anteriormente, pequenas mudanças no posicionamento do objeto de teste dentro da hierarquia do HTML, ou de seu posicionamento em relação aos outros elementos, podem fazer que os localizadores percam a referência a seus objetos de teste.

No entanto, existem maneiras para evitar esses localizadores e, ainda assim, obter a referência precisa dos objetos. Localizadores XPath que usam a estrutura hierárquica são frágeis a mudanças de estrutura, e localizadores que usam propriedades internas são frágeis a mudanças de código HTML. Portanto, a solução é referenciar os objetos por outras propriedades que não sejam frágeis a mudanças na página. Essa ação aumenta o tempo de vida dos casos de testes, solucionando o problema apresentado anteriormente, na Seção 3.1.2.

3.2.5 Planejamento automático dos casos de teste

Para que os casos de testes sejam planejados, é necessário que o engenheiro de teste tenha conhecimento amplo da aplicação sobre testes, o que requer tempo e recursos. Isso se torna difícil em sistemas que evoluem constantemente e é prejudicado caso não haja uma boa comunicação entre a equipe de desenvolvimento e os testes.

O SIATES usa como alternativa o mapeamento automático da aplicação sobre testes. Os fluxos de navegação e execução dos testes são baseados na própria aplicação. Dessa forma, os fluxos estarão sempre atualizados e disponíveis para a equipe de testes, de forma que se torna necessário somente selecionar os fluxos que sejam relevantes para o teste a ser executado.

Capítulo 4

Proposta de Arquitetura

Este capítulo irá detalhar a arquitetura SIATES, que é uma proposta para solucionar os problemas apresentados anteriormente. A ferramenta SIATES realiza a automação de aplicações *web* usando tecnologias de automação de testes e de aprendizado de máquina. A ferramenta foi criada com a utilização da linguagem de programação Java e faz uso de uma interface *web*.

Figura 4.1: Arquitetura do SIATES.

A Figura 4.1 representa a arquitetura geral do SIATES, incluindo suas ferramentas. Essa arquitetura é dividida em três camadas principais: a primeira camada é a camada de apresentação, por meio da qual a equipe de teste irá interagir com o SIATES. A segunda camada é a camada de processamento, responsável pela criação e pelo gerenciamento de casos de testes. Por fim, a terceira camada é a canada de automação, onde será feito o teste da AUT. Tais camadas e componentes são explicados a seguir.

4.1 Camada de apresentação

A camada de apresentação é aquela em que o testador irá interagir com o SIATES. Por meio dessa camada também são visualizados os resultados da execução dos testes. A interface foi criada usando Java *web*, juntamente com *framework* JSF e Primefaces.

4.2 Camada de processamento

A camada de processamento é responsável pela criação dos casos de testes. A camada de processamento é dividida em três submódulos principais: módulo de classificação de mensagens, módulo de classificação de objetos de teste e módulo de geração de caso de teste.

4.2.1 Classificação de mensagens

Este submódulo é responsável por classificar automaticamente as mensagens de *status* de sistema por meio do *framework* OpenNPL. Uma mensagem de *status* de sistema é a mensagem de sucesso ou erro que é exibida pela AUT ao final de uma operação. Por exemplo, se um usuário realiza um cadastro preenchendo todos os campos corretamente, o sistema pode exibir a mensagem de status 'Registro cadastrado com sucesso'.

A classificação das mensagens de sistema é feita em duas categorias: mensagens de sucesso e mensagens de erro. Mensagem de sucesso informa que a operação feita pelo usuário não ocasionou nenhum erro na AUT ou não violou nenhuma regra de negócio do sistema. Mensagem de erro informa ao usuário que algo de errado ao executar a operação na AUT, o que pode ter sido ocasionado por um erro do sistema ou do usuário.

Para fins de reprodução desse experimento, o fator determinante para o corte de repetição de palavras, denominado *cut-off*, foi um, pois foi o número que apresentou uma maior representabilidade dos dados e uma menor quantidade de palavras de baixa qualidade, como pode ser observado na tabela 4.1.

O código usado para a classificação das mensagens de texto está disponibilizado no Anexo B; as amostras das mensagens de erros são apresentadas no anexo A.1; e as mensagens de sucesso no anexo A.2.

4.2.1.1 Amostra de mensagens

Para o treinamento do algoritmo de classificação de mensagens de *status* de sistema, foram usadas mensagens de três sistemas diferentes. As mensagens foram pré-classificadas manualmente entre mensagens de erro e mensagens de sucesso, de modo que ocorreram 914

Tabela 4.1: Relação entre *cutoff* e precisão na classificação de mensagens de sistema

cutoff	**precisão**
1	0.9898
2	0.9812
3	0.9754
4	0.9737
5	0.9712
6	0.9712
7	0.9672
8	0.9661
9	0.9644
10	0.9542

mensagens de erro e 370 mensagens de sucesso, o que totalizou 1.283 mensagens. Para manter a diversidade das mensagens, três sistemas de diferentes domínios foram selecionados, um sistema de gestão orçamentária, um sistema de cadastro de cartório e um sistema de cadastro de documentos, esses sistemas foram usados por usarem os padrões de cadastro e pesquisa cobertos pelo SIATES.

4.2.2 Classificação de objetos de teste

Este módulo é responsável por classificar os objetos de teste obtidos da AUT. Esses objetos precisam ser classificados para serem usados posteriormente na navegação da AUT, na extração de dados e no cadastro de registros nos formulários da AUT. A classificação de objetos de teste foi feita no *framework* Weka[1] usando o seu algoritmo de RNA.

As *features* usadas para serem inseridas nos neurônios de entrada da RNA foram obtidas da estrutura DOM dos objetos de teste, em quatro hierarquias, ou seja, um campo de texto que esteja dentro de um formulário em um objeto de teste 'div', terá a seguinte hierarquia 'body>div>form>input', cada uma dessas variações correspondem a uma das *features*.

No entanto somente a hierarquia do HTML não é o bastante para classificar os objetos de teste, assim, além das variações do *HTML* são consideradas as seguintes *features*.

- *Input* irmão: Esta *feature* é composta pela quantidade de objetos de teste *HTML* do tipo '*input*' que estão no mesmo nível hierárquico.
- Célula irmão: Esta *feature* é composta pela quantidade de células de tabela HTML que estão no mesmo nível hierárquico.
- Menu irmão: Quantidade de objetos de teste de *link* HTML que tenham a propriedade '*href*' e que estejam no mesmo nível hierárquico.

[1] https://www.cs.waikato.ac.nz/ml/weka/

- Contém filho: Se o objeto de teste HTML contêm filhos em sua estrutura hierárquica.
- Contém texto: Se o objeto de teste HTML contém algum texto em sua estrutura.

A rede neural usada foi baseada em perceptron multicamada, contendo uma camada de entrada com 74 neurônios, onde esses neurônios representam cada uma das entradas binárias das *features*, uma camada oculta com 40 neurônios, e uma camada de saída com 3 neurônios. A Figura 4.2, mostra a estrutura neural da rede com a quantidade de neurônios de cada camada, as variações e legendas dos neurônios são detalhados no Capítulo de Anexos, no Anexo D.

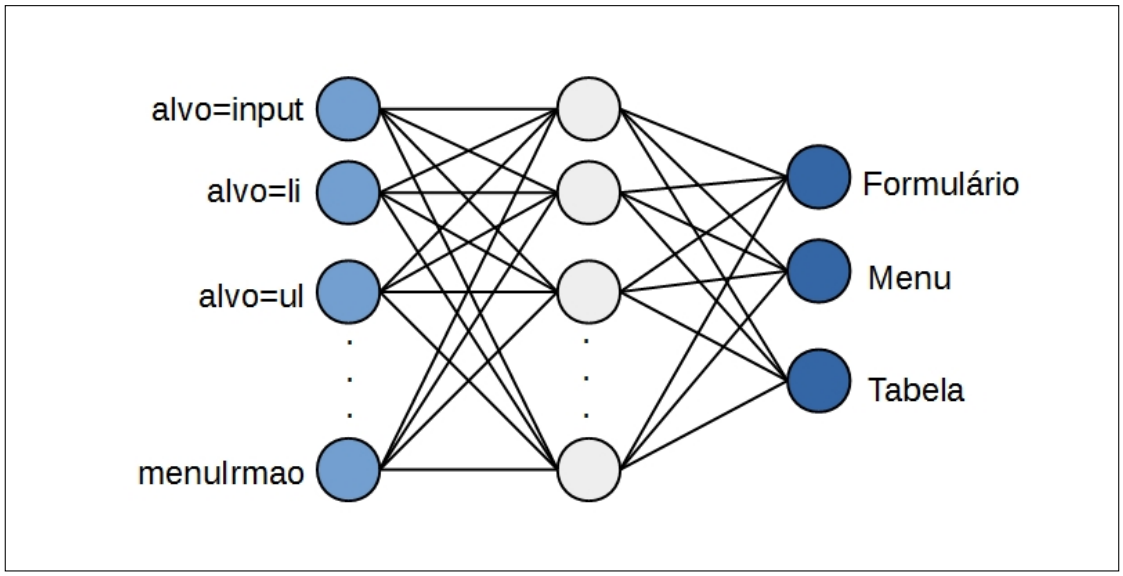

Figura 4.2: Arquitetura da rede neural do SIATES.

Parâmetros como *learning rate*, que é a taxa em que a rede aprende, devem ser definidos. Para determinar quais os melhores valores para os parâmetros de *learning rate* foi realizado um teste onde *learning rate* está entre 0.1 e 0.5 . A Tabela 4.2 mostra a precisão em comparação aos parâmetros citados.

Tabela 4.2: Relação entre *learning rate* e precisão na classificação de objetos de teste.

Learning Rate	**Precisão**
0.1	98.336%
0.2	98.336%
0.3	98.336%
0.4	98.248%
0.5	0.963%

Ao executar tais parâmetros com 500 iterações, não houve mudança de precisão significativa, com exceção quando o *learning rate* é maior que 0.4, que ocasiona diminuição da precisão, a parte disso, nessa quantidade de iterações a precisão não variou do valor de

98.336%, os parâmetros escolhidos para a classificação de objetos de teste foram 0.3 para *learning rate* pois foram os parâmetros que obtiveram o maior nível de qualidade com o menor tempo de processamento.

4.2.3 Geração de casos de teste automatizados

Este módulo é responsável por processar as informações coletadas na AUT e, a partir delas, gerar um caso de teste automatizado. As informações são armazenadas em uma base de dados em grafo denominada Neo4j [2], o qual também é responsável por relacionar alguns dos objetos de teste de interface entre si. Além disso, um Algoritmo Genético (AG) é usado para criar os dados de entrada de teste que são usados no caso de teste.

Os objetos de teste da AUT são extraídos e classificados com uma RNA e armazenados no Neo4J. Esses objetos são identificados pelo seu nome, que é obtido de acordo com a Tabela 3.1. Os nomes dos objetos de teste são comparados para verificar se existe elementos com nomes iguais. Baseado na igualdade entre os nomes dos objetos de teste os elementos são relacionados no banco de dados.

Para compor o restante do caso de teste, os dados de entrada são gerados fazendo uso de um algoritmo genético. Os dados de entrada de teste são criados a partir dos dados extraídos, gerando novos indivíduos no algoritmo genético. Cada um dos genes dos indivíduos da geração anterior é selecionado randomicamente, e a partir desses genes, um novo indivíduo será criado.

Para garantir que a mutação não diminua o *fitness* dos indivíduos, a mutação é controlada por meio de uma variável que determina o quanto o valor pode sofrer mutação. Essa variável denomina-se taxa de uniformidade que é definida como 1, outro parâmetro que têm influência no processo de geração de novos indivíduos é o tamanho do torneio, que determina quantos indivíduos serão selecionados para competir para serem usados em uma nova geração, o tamanho do torneio usado aqui foi 1. A taxa de uniformidade influencia a mutação randômica do indivíduos após ele ser selecionado no SIATES ele é definido como 0.1.

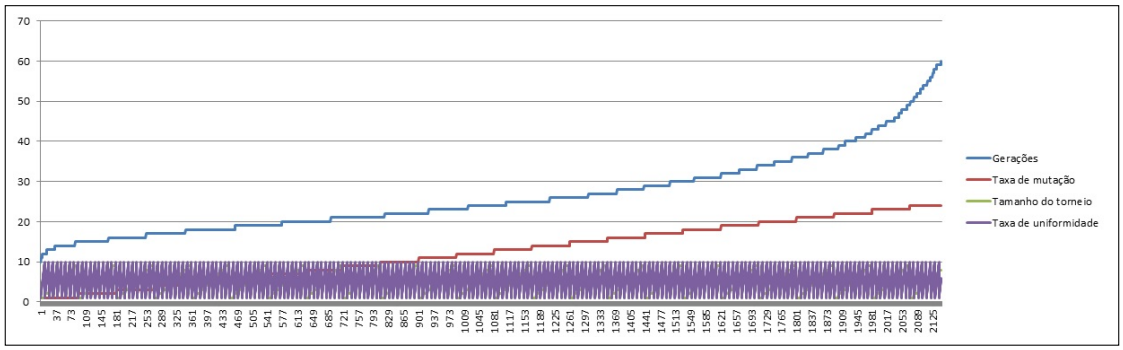

Figura 4.3: Relação entre quantidades de gerações necessárias para criar um dado de entrada de teste e os parâmetros de um algoritmo genético.

[2]https://neo4j.com

A Figura 4.3 mostra a relação entre a quantidade de gerações que são necessárias para gerar um dado de entrada de teste, quanto menor a quantidade de gerações melhor. Em um total de 2147 testes com parâmetros variados, somente uma combinação de parâmetros foi capaz de gerar um valor em 10 gerações, que são os parâmetros usados nos algoritmos genéticos do SIATES.

4.3 Camada de automação

A camada de automação é responsável por fazer a interação com o navegador. Essa camada recebe o caso de teste que é composto por uma série de etapas a serem executadas. A interação do SIATES com o navegador é feito por meio de uma ferramenta de automação de testes denominada Selenium [3], que é usada também para a extração de conteúdo.

4.4 Fluxo de execução do SIATES

Nesta Seção, será explicado o fluxo de funcionamento do SIATES. O fluxo de apresentação dos tópicos usados é refletido pela ordem em que os componentes são executados ao gerar e executar os casos de testes automatizados.

Aqui, será detalhado o modo de interação com as páginas de teste; a localização e a identificação dos objetos de teste em tela usando RNA, a criação de novos dados de entrada de teste, o mapeamento de uma aplicação usando Bancos de dados em grafo e a identificação das mensagens usando PLN.

4.4.1 Extração de dados da aplicação sobre testes

Nesta Subseção, será explanado o processo de extração de informações da AUT. A extração de informação é o processo de obtenção dos dados relevantes para a automação de testes, sendo eles, os caminhos de execução, campos de entrada e objetos de teste de saída de dados.

A extração de informações é feita com a ferramenta Selenium, por meio da XPath [Anton 2005], que também é usada para a execução dos testes.

4.4.2 Interpretação dos objetos de teste da AUT

A fase de interpretação dos objetos de teste busca identificar os tipos de objetos obtidos na fase anterior, buscando diferenciá-los em três categorias, a categorização dos objetos de

[3] http://www.seleniumhq.org

teste é feita para mapear os dados do sistema. Isso é feito por meio de uma RNA utilizando o Weka.

4.4.3 Classificação dos objetos de teste da AUT

A fase de classificação dos objetos de teste na AUT busca identificar os tipos de objetos de teste obtidos na fase de extração, classificando-os em três categorias: campos de entrada, campos de saída e objetos de teste de navegação, a classificação dos objetos de teste é feita com uma RNA.

Campos de entrada são objetos de teste onde o usuário pode informar dados de entrada ao sistema, incluindo campos de texto e *comboboxes*, campos de saída são objetos de teste onde o sistema informa dados de entrada ao usuário, como por exemplo, tabelas e listas, elementos de navegação são usados para levar o usuário a diferentes partes do sistema, como menus.

Existem quatro objetos de teste essenciais para a criação dos testes, navegação no sistema e interpretação do *feedback* da AUT. Esses elementos, que são extraídos com Selenium em conjunto, estão exemplificados na Tabela 4.3.

Tabela 4.3: Elementos extraídos para a geração de casos de teste.

Elemento	Descrição
Elementos de navegação	Representam os menus da aplicação sobre testes, usado para acessar as diferentes partes do sistema.
Formulários	Formulários são onde o usuário irá prover dados ao sistema. Compostos por um conjunto de campos de entrada.
Estruturas de dados	Elementos de interface que permitem a aplicação sobre testes exibir informações ao usuário final. Tais informações geralmente refletem tabelas de bancos de dados que recebem tratamentos ou não antes de serem exibidas ao usuário final.
Mensagens de sistema	A forma que a aplicação sobre testes tem para informar ao usuário o status de uma operação realizada.

4.4.4 Navegação na AUT

Esse processo é feito para obter a estrutura navegacional AUT. Os dados obtidos são usados para o planejamento automático dos casos de testes no que se refere aos caminhos que são usados para compor os testes. Para fazer a navegação no sistema, e interagir com os objetos de teste é usada o *framework* Selenium.

Os objetos de teste são analisados para identificar onde o comando *hover* ou *click* será usado. Isso é feito para que os *links* possam estar visíveis e acessíveis antes de serem clica-

dos.

4.4.5 Extração de informações dos objetos de teste de saída de dados

Essa fase da extração busca obter os valores de saída do sistema, para que possam ser diretamente usados nos testes nas fases posteriores ou usados como parâmetros de comparação na produção de novos valores de teste usando AG, conforme demonstrado na Seção 4.6.5. O uso dessa técnica é justificado pela diminuição dos custos e do aumento da velocidade e da produtividade da criação de testes.

Existem outros estudos prévios que realizaram a extração de entidades de tabelas HTML [Crescenzi et al. 2001], [Wong et al. 2009], [Tengli et al. 2004], [Dalvi et al. 2012], utilizando diversas técnicas, como AM ou AG para a extração de dados de tabelas e outras estruturas. A solução apresentada usa a mesma técnica proposta no estudo [Purnamasari et al. 2013] para extração dos dados de uma tabela HTML, o que é feito por meio da contagem das *tags*.

4.4.6 Identificação de objetos de formulário

O propósito da identificação dos objetos de formulário é definir quais as interações são usadas para cada campo. Na automação de testes, as ações são determinadas pelo testador para compor os casos de testes. A nomeação desses campos é feita priorizando-se a fácil identificação do campo em código em relação ao campo em tela de acordo com o que já foi apresentado.

Os campos extraídos seguem os padrões já apresentados em estudos anteriores [Bolin and Miller 2005], [Glover et al. 2002], de forma que os objetos de teste apresentados pela AUT podem ser facilmente identificados pelo usuário final. A extração de cada objeto é exemplificada a seguir:

1. Links: A nomeação de *links* é feita de acordo com o seu conteúdo, ou seja, o texto que está contido no *link*. O código-fonte demonstrado no Algoritmo 4.2 usa o texto puro entre as *tags* da linha um e três.

Algoritmo 4.1 Exemplo de nomeação de *link* HTML

```
1: < ahref = "servicoLogar.jsp»Logar < /a >
2: Logar
3: < /a >
4: link.getText()
```

O algoritmo 4.2 é um exemplo de *link* onde o usuário irá clicar para navegar até a página de login; o seu nome será armazenado usando um método Java do *framework* Selenium, linha 4.

No entanto, existem casos em que a imagem não contém nenhum valor em texto, o que ocorre quando imagens são usadas. Quando isso ocorre, a propriedade 'alt' da imagem é preferencialmente usada para a identificação do objeto.

2. Botões: o usuário identifica um botão pelo texto visível dentro dele. Existem três tipos básicos de botões em HTML e, para cada um deles, existe uma abordagem para extrair o seu nome.

   O algoritmo 4.3 exemplifica três modelos de botões que, apesar de terem implementações diferentes, são renderizados da mesma forma para o usuário.

Algoritmo 4.2 Exemplo de nomeação de botão HTML

1: $< buttonname = "Login»Logar < /button >$
2: $< inputtype = "button" value = "Logar» < /input >$
3: $< inputtype = "submit" value = "Logar» < /input >$

   Para o primeiro exemplo, linha 1, no algoritmo 4.3, o conteúdo entre as *tags* será extraído de modo similar ao método de extração de nome de *links*. Nos dois últimos exemplos, o nome é extraído das *tags* '*value*'. Alternativamente, existe a possibilidade de extrair os valores da *tag* '*name*' ou '*alt*', caso o botão não contenha nenhum valor em texto para *value*.

3. Campos de texto, senha, *textArea* e *comboBoxes*: tais campos geralmente não contêm um objeto de teste identificador embutido em seu código-fonte, no entanto sua identificação é feita por um objeto de teste externo.

   Para esses e outros objetos de teste, um objeto do tipo '*label*' é usado próximo ao campo que o identifica. Existem múltiplas abordagens usadas para nomear um objeto desse tipo e associá-lo a sua *label* correspondente.

4. *CheckBoxes* e *radiobuttons*: Podem ser identificados pela sua propriedade *value* e podem ser nomeados usando a mesma técnica para nomeação de campos de texto. Apesar de serem usados em conjunto, cada uma das opções referenciam valores diferentes, por isso são identificados individualmente.

4.4.7 Mapeamento de objetos de saída de dados, formulários e entidades na AUT

Mapeamento é o processo de representar a AUT em forma de grafo. Cada um dos registros é representado como nós dentro da base de dados em grafo Neo4J[4]. O SIATES irá fazer a leitura dos nós de forma a identificar relacionamentos entre eles, processo que será baseado nas similaridades de seus identificadores, relacionando os objetos de testes pelos seus nomes.

[4]https://neo4j.com

Menus são armazenados de forma a manter a sua estrutura hierárquica original. Os nós em que são transformados são 'Módulos' e 'Páginas'. Módulos são nós usados para representar a estrutura hierárquica HTML, que pode ser organizado em uma ou múltiplas páginas dentro de um sistema.

4.4.7.1 Criando caminhos de acesso a página

A partir das informações persistidas no banco de dados em grafo, é definido o fluxo de execução de teste automatizado. Dentro do banco de dados, os módulos e as páginas da AUT estão persistidos em forma de nós, que estão relacionados entre si. Dentro desses nós, existem informações que são usadas pela ferramenta de automação para acesso a esses objetos por meio da interface HTML. Com essas informações, são obtidos os fluxos de acesso às funcionalidades do sistema.

4.4.8 Definição de preenchimento de formulário

A fase de definição de preenchimento define a ordem em que os campos são preenchidos, normalmente seguindo o padrão de fluxo usado por um testador manual, ou seja, ao testar um formulário de preenchimento, os campos são normalmente preenchidos sequencialmente de cima para baixo e da esquerda para a direita, caso haja múltiplas colunas. Por fim, o formulário é submetido à operação 'Salvar' ou 'Editar', por exemplo.

4.4.9 Definição de ações de preenchimento

As ações de preenchimento definem quais as ações são executadas de acordo com os diferentes tipos de campos de entrada. A partir de propriedades armazenadas no objeto, é possível obter o tipo de objeto de teste (por exemplo, botão, *link*, campo texto), o que, por sua vez, é usado para definir as ações adequadas para cada tipo de objeto de teste. A Tabela 4.4 mostra a relação entre os tipos de objetos de teste e suas ações correspondentes.

Tabela 4.4: Detalhamento dos tipos de objetos de teste

Tipo	Elemento	Ação
input type submit,input type button,button	Botões	Clique
a *href* ,objetos de teste com *onClick* ou *onHover*	Link	*Hover*, ou Clique
Input type text, password, textArea	Caixa de texto	Preencher
Select	*ComboBox*	Seleção(Um clique para abrir o objeto de teste e outro para selecionar seu valor)
input type checkbox	*CheckBox*	Clique, de forma a definir o estado final do objeto
input type radiobutton	*RadioButton*	Clique, em um dos objetos de teste definidos dentro do grupo

4.4.10 Criação de valores de teste

A criação de valores de teste representa uma parte significativa do planejamento de testes, o que justifica os esforços feitos por outros autores para automação da tarefa. A abordagem usada gera uma nova população de dados de testes baseados nos existentes por meio de um algoritmo genético, o que faz que o teste não tenha de ser executado para que os novos dados sejam gerados.

Existem estudos que já usaram de AG para gerar valores para teste [Pargas et al. 1999], [Bolin and Miller 2005], [Girgis 2005], [McMinn 2004]. No entanto, o teste da própria aplicação sobre testes é usado para avaliar a qualidade do dado, já que há um problema que existe ao testar sistemas *web* que não é tão evidente ao fazer testes unitários. Executar a aplicação sobre testes somente para avaliar um valor usa muitos recursos. Para que o sistema *web* processe uma informação, os seguintes passos são executados:

1. O sistema tem de estar disponível para que os dados possam ser inseridos no sistema, ou seja, o formulário de entrada tem de estar totalmente carregado. Os campos que usam valores predefinidos, como *comboBoxes*, tem de ter os seus valores carregados.

2. Após o preenchimento dos campos, uma requisição terá de ser feita ao sistema, de forma que todos os valores inseridos pelo usuário sejam enviados ao sistema para processamento por meio da internet.

3. Os valores são validados e processados, usando as regras de negócio do sistema. É possível que o sistema tenha de fazer acesso a recursos externos para a validação, como bancos de dados, *web-services* ou outros módulos do sistema.

4. Em caso de sucesso, nas operações que requerem a persistência de dados, o sistema usa uma conexão com o banco de dados para salvar os valores necessários.

5. O sistema atualiza o formulário para exibir o status atual da operação com uma mensagem de sucesso ou erro. Ao atualizar a página, mais recursos são consumidos, pois é necessário que o navegador baixe e renderize a mensagem de sucesso ou erro.

Fazer o uso de AG para testar os valores diretamente pela interface de aplicação seria muito demorado em comparação a outras metodologias, como geração de dados aleatória [Korel 1990], [Korel 1996]. Mesmo que a operação demore meio segundo para executar o teste, a quantidade grande de testes necessários para evoluir uma população faria que a operação se tornasse ineficiente.

A solução encontrada nessa abordagem foi usar os valores mapeados pelo sistema, ou seja, o teste realizado é feito comparando-se um determinado valor com os valores que já foram inseridos no sistema, ao invés de usar o sistema.

Antes de criar os novos valores, é necessário normalizar os valores existentes, o que é feito para a operação de *crossover* seja mais consistente. Primeiramente, o tamanho das *strings* é padronizado por meio da obtenção do maior número de indivíduos que contenha o mesmo número genes. Esse grupo será usado como base para determinar as características que os valores de diferentes tamanhos terão de ter.

O algoritmo de cálculo de *fitness* é similar ao algoritmo usado para a padronização de tamanho dos indivíduos. No entanto, será feito uma verificação para que o sistema não crie valores que já existem na base de comparação, o que, na prática, evita erros de validação provocados pela entrada de dados repetidos.

4.5 Execução de casos de teste

A arquitetura proposta para o fluxo de execução de um caso de teste é apresentada na Figura 4.4. A figura representa o fluxo comum de execução de um dos comandos do caso de teste. Isso é feito para evitar o uso de localizadores ruins e, consequentemente, evitar a quebra do teste quando ele for rodado novamente. Detalhes de cada passo são exemplificados nas subseções seguintes.

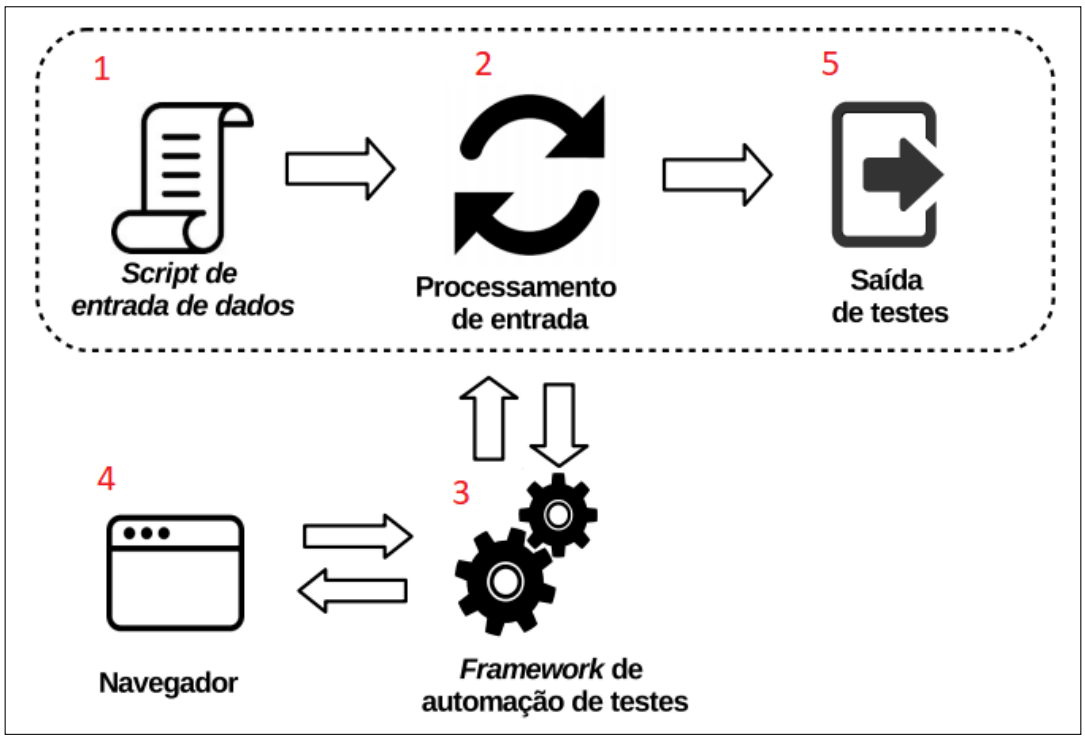

Figura 4.4: Fluxo de execução de um caso de teste no SIATES.

4.5.1 *Script* de entrada de dados

A linguagem de entrada de dados usada para a criação do caso de teste tem influência direta na produtividade, como discutido anteriormente. Fazer uso de uma estrutura simplificada, em que é possível escrever o caso de teste em diversos editores, aumenta o interesse externo e possibilita a automação por pessoas que não possuam conhecimento em programação.

A entrada de dados na arquitetura proposta é realizada através do *framework* FIT [5]e o formato é o de tabelas. O propósito original do *framework* FIT é realizar testes unitários no código-fonte da aplicação, porém, no contexto dessa solução, foi adaptado para a automação de testes de interface. A ferramenta utiliza tabelas para realizar a entrada de dados, de forma a executar arquivos de código-fonte, usando parâmetros e comparando a saída com todas as informações necessárias contidas nas células das tabelas.

4.5.2 Processamento de entrada

A camada intermediária para o processamento de entrada de dados converte o *script* juntamente com os comandos pré-configurados para executar o teste [Little and Miller 2006]. A parametrização permite atender às necessidades da equipe e sua evolução de acordo com a aplicação sobre os testes, resultando em uma redução do tempo de treinamento. A arquitetura

[5]http://fit.c2.com

de processamento de comandos é dividida em três camadas básicas, conforme apresentado na Figura 4.10 e detalhado na Tabela 4.4.

Tabela 4.5: Descrição das camadas.

Camada	Descrição
Camada de *script*	Camada onde é configurável, os comandos em alto nível, informados pelo testador inclui as palavras chaves, que são usadas na automação.
Camada de automação	Onde os comandos interagem, com a página *web* são, chamados, baseado nas palavras chaves mapeadas na camada de *script*. Comandos como: preencher, clicar e checar, são executados imitando as ações de um testador manual.
Página *web*	Representação dos objetos de teste em HTML esses objetos são mapeados para os comandos de automação onde cada comando têm um tipo compatível de objeto de teste.

Tal processamento permite que uma linguagem mais fácil seja usada, o que mitiga problemas relacionados à complexidade e à manutenção, discutidos anteriormente nas Sessões 3.2.2, 3.2.2.2 e 3.2.3.

4.5.3 *Framework* de automação de testes

Após os comandos serem pré-processados, eles são repassados para o sistema de execução do Selenium, de forma que os objetos de testes sejam dinamicamente localizados usando a informação do caso de teste. A ferramenta de automação traz informações de teste que podem ser úteis na execução, informa o status que irá ser exibido ao usuário bem como executa todos os comandos no navegador.

4.5.4 Navegador

O navegador é usado no mesmo servidor da aplicação, com suporte para um ambiente gráfico. Para melhor performance e precisão, o navegador deve estar configurado para não salvar *cookies*, senhas e outros dados de navegação.

4.5.5 Saída de testes

O resultado de execução do caso de teste será armazenado em um dataset de forma a ser usado posteriormente para compor os casos de teste. O status dos comandos é armazenado individualmente.

No tempo de execução, é possível acompanhar o *status*, de forma que ele possa ver o estado atual da aplicação sobre testes sem a possibilidade de que o usuário interaja, por acidente, com a aplicação. Os detalhes dos testes de execução são exibidos por comandos.

4.6 Interpretação de mensagens de *status*

A interpretação automática das mensagens de texto tem o papel de detectar se uma funcionalidade da AUT resultou em erro ou sucesso. O sucesso ou erro da aplicação deve ou não acontecer, o que depende diretamente do contexto e dos parâmetros passados ao sistema. Automatizar essa tarefa auxilia o testador, de modo que ele somente terá de validar se o status resultante da execução estiver de acordo com o que é esperado.

A solução teve de ser treinada para reconhecer diferentes mensagens de sistema. O primeiro conjunto de dados contém mensagens de erro e sucesso dentro do contexto da aplicação de teste e foram categorizados em erro e sucesso, respectivamente. O conjunto é usado, posteriormente, dentro do SIATES para detectar futuras mensagens de sistema e determinar o erro ou sucesso da operação executada. A Tabela 4.6 exemplifica algumas das mensagens usadas e seus respectivos status.

Tabela 4.6: Amostra de mensagens de *status* de sistema para treino de modelo de classificação.

Status	Mensagem
Sucesso	Cadastro realizado com sucesso.
Sucesso	Registro salvo com sucesso.
Sucesso	Romaneio recebido com sucesso.
Erro	Horário de funcionamento inválido.
Erro	Impossível excluir este registro. Ele está vinculado a outro registro
Erro	Os campos devem ser preenchidos

No segundo conjunto de dados, foram usadas somente as mensagens de erro que, novamente, foram classificadas em diferentes categorias. O reconhecimento dessas categorias será usado posteriormente na execução dos testes, caso o fluxo de teste resulte em um erro de sistema. A categoria é usada para auxiliar a tomada de decisão envolvendo o fluxo e os dados usados na próxima iteração de teste, a fim de resultar em uma operação realizada com sucesso.

Capítulo 5

Experimentação e análise dos resultados

Este Capítulo detalha o processo de experimentação utilizado para comprovar o sucesso dos métodos empregados neste trabalho. Tais experimentos foram executados com a utilização de dados reais bem como foi criada uma aplicação sobre testes com o propósito de testar o algoritmo de identificação de objetos de teste em página.

O Capítulo está organizado de forma a representar a ordem de execução do sistema em si, sendo os experimentos compostos por localização de objetos de teste dentro de uma página, usando RNA com uma base de dados de múltiplos *templates* HTML; mapeamento de sistema, com a utilização conjunta de automação, a fim de validar a possibilidade de mapeamento entre meios de entrada e saída em um sistema; categorização das mensagens de texto, usando um algoritmo de classificação; criação de novos valores baseada em valores já obtidos usando AG; e, por último, a execução e o resultado dos casos de teste automatizados criados pelo sistema.

5.1 AUT usada para experimentação do mapeamento automático de sistema

Realizou-se um experimento criando-se uma AUT, com o propósito de ser usada como base para a execução dos testes de navegação e para a realização do mapeamento.

A aplicação sobre testes usada para a validação da ferramenta é composta pelos módulos Aluno, Carro, Empresa, os quais representam diferentes partes de um sistema que trabalham em conjunto.

A estrutura hierárquica de navegação do experimento é dividida entre módulos e páginas: os módulos são usados somente para a organização, ao passo que as páginas redirecionam o usuário para os diferentes pontos do sistema, exemplificado na Figura 5.1.

É importante observar que a ordem dos objetos de teste no menu não representa a ordem em que os testes devem ser executados, ou seja, para que o sistema cadastre um usuário com

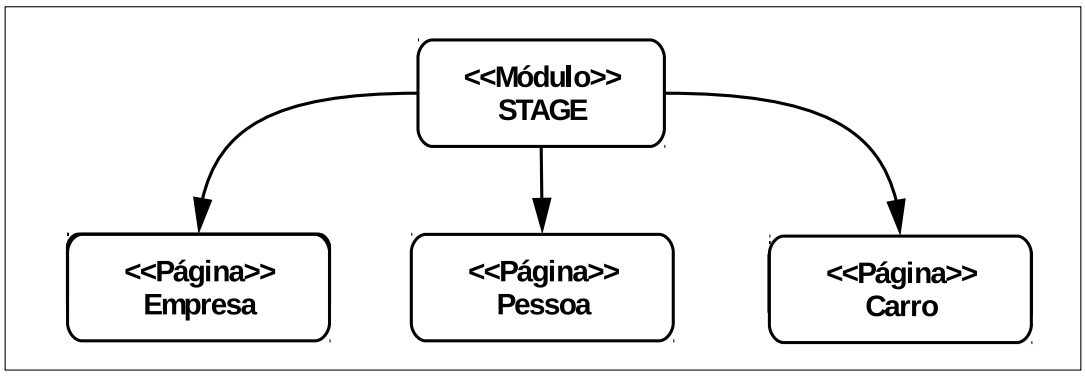

Figura 5.1: Organização hierárquica de páginas dentro da estrutura de menu.

sucesso, deve-se cadastrar, primeiramente, um carro e uma empresa.

5.2 Reconhecimento de objetos de teste

Este experimento avalia a capacidade do SIATES de identificar diferentes objetos de teste em uma página HTML. RNA mostraram-se eficientes em métodos de classificação [Specht 1990], inclusive para conjunto de dados pequenos [Hepner 1990].

Para a realização do experimento de reconhecimento de objetos de teste, foram coletados *templates* HTML de múltiplas fontes que disponibilizam exemplos gratuitamente, como Open Designs [1] e Free CSS [2].

O código HTML foi extraído de cada um dos exemplos mencionados e separado em três categorias. Em cada uma das categorias, a árvore DOM e outras informações foram extraídas, obtendo-se o objeto de teste alvo e sua estrutura correspondente.

Elemento-alvo é o objeto de teste HTML onde a ação aplicada, ou seja, os *links* de um menu, os campos de entrada de um formulário e os campos de saída de uma tabela. A Tabela 5.4 detalha os dados usados para o treino da rede neural.

Tabela 5.1: Quantidade amostras.

Elemento	Estrutura de dados	Elemento alvo
Menu	700	Link, botão, elemento clicável
Estrutura de dados	700	Texto de célula de tabela
Formulário	700	Elemento de entrada de dados

Os objetos foram identificados com base na sua *tag* e, para o reconhecimento, foi considerado o elemento-alvo, que está quatro níveis acima em relação aos objetos da árvore . A

[1] disponível em : http://www.free-css.com/ Acessado em : 2016-09-13.
[2] http://www.opendesigns.org/website-templates/ Acessado em : 2016-09-13

extração dos itens foi feita com a ferramenta JSoup [3], de forma a localizar o objeto-alvo e, a partir dele, foram obtidos os pais dos objetos de teste, os quais foram armazenados em memória. A Figura 5.5 mostra a hierarquia extraída a partir do elemento-alvo. O elemento-alvo representa o objeto de teste em que o sistema de automação irá agir, ou seja, o botão que será clicado ou o campo de texto que será preenchido; o elemento-pai representa o nó que está imediatamente acima do elemento-alvo; o elemento-avô representa o objeto de teste que está imediatamente acima do elemento-pai. O elemento-bisavô é o objeto de teste que está imediatamente acima do elemento-avô.

Tabela 5.2: Exemplificação de hierarquia extraída a partir do elemento alvo.

Elemento bisavô	Elemento avô	Elemento pai	Elemento alvo
div	table	tr	td
from	div	div	select
div	ul	il	link

Além da estrutura hierárquica, outros dados foram analisados, pois existem casos em que a estrutura hierárquica não é suficiente para realizar a classificação de um objeto de teste, ou seja, somente com a hierarquia uma lista poderia ser confundida com um menu; uma tabela usada para posicionamento dos objetos de teste em tela poderia ser confundida com uma tabela para a exibição de valores; e um campo de pesquisa na barra superior de um *site* poderia ser confundido com um campo de formulário.

Tabela 5.3: Precisão de classificação por meio de RNA.

TP	Classe
0,917	Menu
1,000	Tabela
0,997	Formulário
0,993	Outro

Como pode ser observado na Tabela 5.6, há uma boa taxa de sucesso na classificação de tabelas, o que se deve ao fato de que as *features* extraídas dos componentes de tabelas não são compartilhadas pelas features dos outros componentes. As propriedades da tabela 5.6 estão explanadas a seguir.

- *TP: True Positive* instâncias corretamente classificadas, quanto mais próximo de 1 melhor

[3]https://jsoup.org

A Tabela 5.7 determina a precisão obtida com o processo de classificação por meio de uma matriz de confusão. A matriz de confusão indica a medida efetiva do método de classificação, mostrando o número de instâncias em contrapartida às classificações preditas para cada classe (Monard and Baranauskas 2003).

Tabela 5.4: Matriz de confusão da classificação.

a	b	c	d	Classificado como
3185	0	0	290	a = menu
0	3734	0	0	b = tabela
0	0	2338	8	c = formulário
0	0	26	3480	d = outro

5.3 Geração de dados de teste

Este experimento busca validar a qualidade de dados de teste gerados automaticamente com a utilização de AG. O objetivo é gerar dados sintéticos que tenham as mesmas propriedades presentes nos dados reais e que não sejam iguais a nenhum dado existente no sistema. A validação usada ao final do experimento irá simular a validação feita por um campo de entrada de sistema.

Os tipos de dados aqui usados são 'E-mail', 'Nome' e 'Data', pois esses dados semi-estruturados são encontrados em sistemas de cadastro *web* e servem como ponto de partida para a geração de variáveis mais complexas, objeto de pesquisa de trabalhos futuros.

O AG usado para a criação dos valores usa uma abordagem de torneio e mutação, com a taxa de mutação de 0.01 pois é a melhor taxa de mutação segundo os testes apresentados no Capítulo 4. A taxa de mutação é o operador usado para diversificar uma geração da outra, e um valor baixo é usado para que os ganhos de *fitness* obtidos na geração não sejam perdidos. Se for utilizado um valor mais baixo que esse,o algoritmo ficaria preso em *local optima* com mais frequência [Nidhra and Dondeti 2012].

A seleção em torneio envolve a seleção de um indivíduo entre uma população de indivíduos, sendo o vencedor desse torneio usado para o cruzamento dos indivíduos da população. A quantidade de indivíduos selecionada no torneio foi 5, pois uma quantidade maior que essa representaria um grande número de elementos com baixo *fitness*, com possibilidade de entrar em uma próxima geração.

Tabela 5.5: Detalhamento de amostra de dados.

Dado	Padrão	Descrição
Data	Um número de 1 a 31, um número de 1 a 12,e um número de quatro dígitos positivos.	O uso desse dado no experimento é usado para testar a capacidade do sistema a lidar com números. Onde diversos fatores devem ser analisados, por exemplo, o mês de fevereiro ser usado com o dia 31.
E-mail	Uma sequencia de carácteres, não iniciada por um nome, seguida por um carácter '@' e por um domínio de e-mail válido.	Esse dado serve para validar a capacidade do sistema a gerar dados que tenham partes fixas, ou seja, o domínio de 'E-mail'.
Nome	Uma cadeia de caracteres intercaladas por espaços	O uso desse dado é importante para validar o tamanho de um texto gerado em relação a sua amostra.

A implementação do algoritmo nesse experimento é feita em Java, sendo executada em um computador com 8 gigas de memória RAM e processador i5.

O algoritmo para cada valor irá rodar com um número fixo de 50 gerações ou até atingir um padrão máximo de qualidade. Esse padrão de qualidade é atingido quando há 5 gerações subsequentes sem melhoras ou quando se identifica o indivíduo de maior *fitness* entre as 50 gerações. A Tabela 5.9 detalha os resultados da execução das tabelas e as variáveis serão explicadas a seguir.

Tabela 5.6: Detalhamento das variáveis de execução do algoritmo genético.

Dado	Erro	Tempo de execução média (ms)
E-mail	23	3188
Nome	0	2293
Data	7	650

O Erro especificado na Tabela 5.6 se refere a quantidade de variáveis geradas com erro em um total de 100 execuções,

A Figura 5.2 mostra a evolução da qualidade do valor de teste em relação a quantidade de gerações, é possível observar que a qualidade aumenta consideravelmente nas primeiras gerações, e que estabiliza rapidamente após esse ponto.

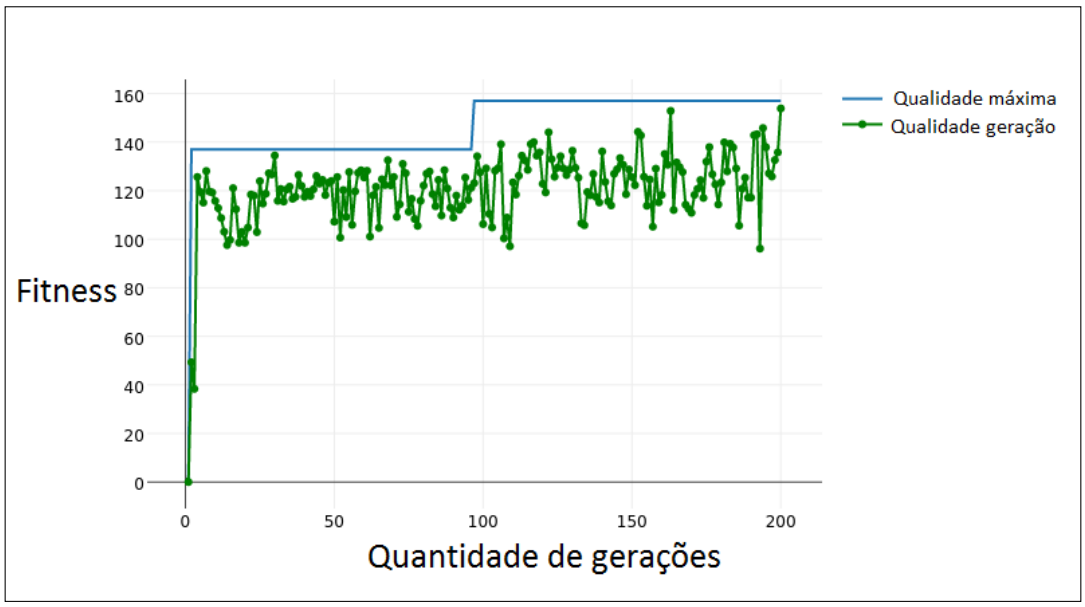

Figura 5.2: Evolução de qualidade por geração.

A massa de dados usada para gerar os E-mails foi divida em três domínios: 50com o domínio de ‘Gmail‘; 25% com domínio de ‘yahoo‘; e 25% com o domínio de ‘hotmail‘. Antes de executar o AG, a massa de dados foi normalizada de forma a selecionar o tamanho do dado do teste mais predominante, ou seja, nesse caso específico, os E-mails com a mesma quantidade de caracteres que aparecem com mais frequência são selecionados para serem usados no AG. Finalizada a execução do AG, foi observado erro em 23 de cada 100 elementos no campo, sendo considerados uma taxa de erro razoável para essa aplicação. Os dois dos tipos de valores gerados com erro. O problema é ocasionado pela diferença de posição do caractere ‘@‘, resultante da discrepância entre o tamanho dos domínios.

Para a criação de dados de data, foram gerados 700 valores no formato DD/MM/AAAA. Entre os períodos de 01/01/2015 a 01/01/2016, houve uma taxa de erro de 7 em 100 elementos gerados. Existem duas causas distintas que provocam esses erros: datas com números acima de 31 e meses que não contêm o dia 31, mas que contabilizam esse dia. Entre os 7 valores errados, 5 deles foram ocasionados por números inválidos no campo mês.

As taxas de erro foram diminuídas, estipulando-se um *fitness* mínimo para que o dado fosse utilizado em outra geração, assim, caso o dado não consiga evoluir em um número preestabelecido de gerações e não contenha uma quantidade mínima de qualidade, o algoritmo será executado novamente.

Devido ao fato do SIATES usar caracteres randomizados baseada na massa de dados, há uma menor variação de caracteres. Como é possível observar, a variação total e média tem influência direta no tempo de execução do algoritmo. A alternativa padrão é usar uma quantidade maior de caracteres que contemplem os diversos problemas. Embora letras, números e símbolos especiais resultem em 256 caracteres, a alternativa apresentada reduz a variação e aumenta a velocidade de execução. No caso da geração de data, o tempo de execução é

reduzido em 95,4%.

5.4 Resultados de navegação, relacionamento e plotagem

Com base na aplicação sobre testes já apresentada, foi realizado um experimento para medir a capacidade da ferramenta de extração, do relacionamento de informações. Esses dados são necessários para a criação de casos de teste e de automação. As imagens apresentadas nessa Subseção foram obtidas através do console de visualização de grafos da ferramenta Neo4J.

5.4.1 Navegação e extração

Na primeira fase, o SIATES lê os objetos de teste de navegação extraído do menu na fase de reconhecimento de objetos e, a partir desses objetos de teste, um acesso é realizado em cada página, a fim de extrair outras informações. A Figura 5.6 mostra o relacionamento entre os objetos de menu. As figuras deste padrão mostram o relacionamento dos dados por meio da interface *web* disponibilizada pela ferramenta Neo4j.

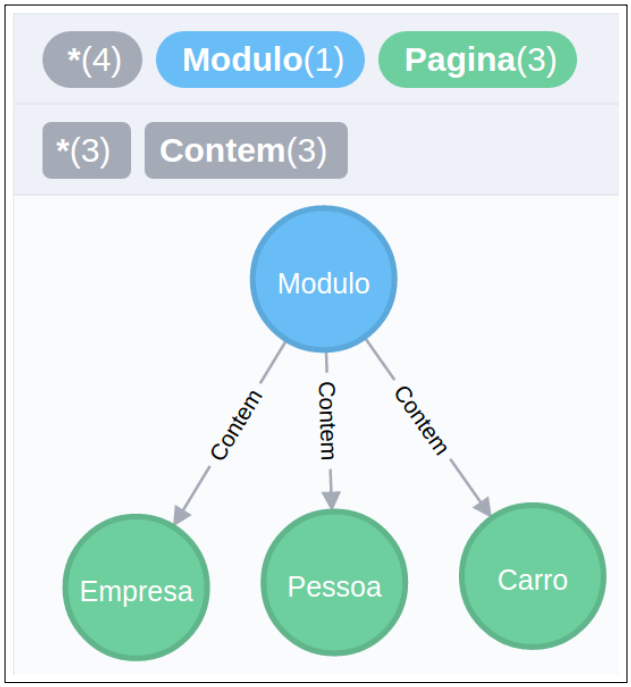

Figura 5.3: Estrutura em grafo de menu extraída da AUT.

Os objetos de teste da página são usados para o acesso ao resto do sistema. O sistema irá entrar de página em página e fazer o reconhecimento de objetos de ‘Formulário‘ e ‘Estrutura de dados‘. Os nomes dos valores de saída são extraídos diretamente do cabeçalho da tabela da estrutura de dados e são associados aos seus respectivos valores. A partir do registro ‘Formulário‘, os ‘Campos de entrada‘ são retirados, já que serão usados em fases

posteriores para receber os valores de teste. Do formulário é extraído um ou mais registros de ‘Funcionalidade‘, que representam ações a serem realizadas naquele formulário.

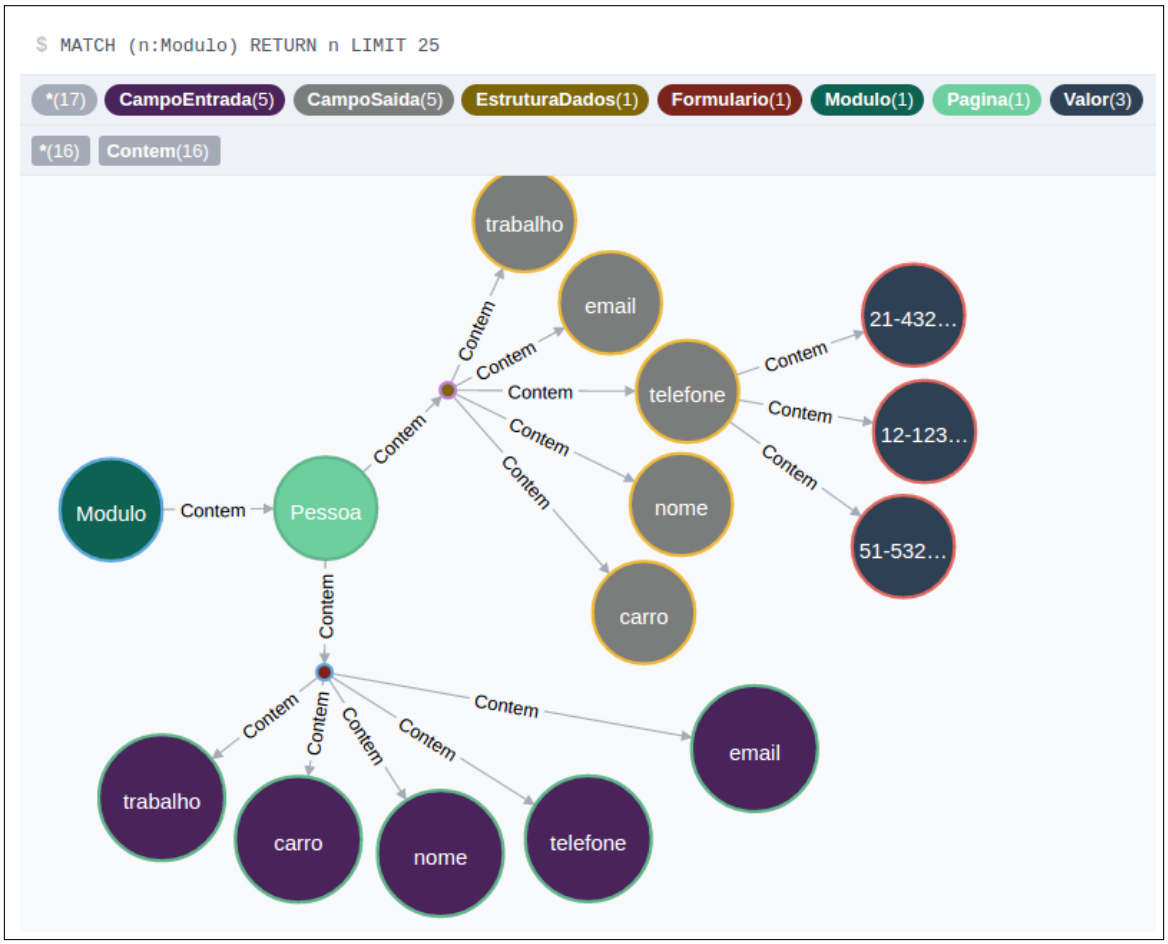

Figura 5.4: Estrutura de valores dentro dos formulários e estruturas de saída.

A Figura 5.6 mostra o ponto inicial, que é o nó de nome ‘Modulo‘, representado na Figura como um círculo verde escuro. A seta ligada ao nó ‘Pessoa‘ representa uma relação entre os dois registros, ou seja, o nó ‘Módulo‘ contém um nó ‘Pessoa‘, que, por sua vez, representa uma página de nome ‘Pessoa‘ em um submenu ‘Módulo‘.

A página pessoa representada pelo círculo verde claro ainda na Figura 5.7 contém um formulário que, por sua vez, contém as entradas de texto representadas pelos círculos de cor roxa. A tabela de pesquisa da página é representada pelos nós de cor cinza, em que cada um contém múltiplos valores, representados pelos nós de cor azul.

5.4.2 Relacionamento de dados

Este experimento irá medir a capacidade do SIATES de identificar relacionamento entre os diferentes objetos de teste em uma AUT, de modo que sua função principal é determinar a origem e a entrada de dados dentro do sistema.

Essa fase permite detectar os diferentes relacionamentos entre os dados do sistema e sua função principal é determinar as origens e as entrada dos dados dentro do sistema e, então, relacioná-las.

A primeira relação é feita entre valores de entrada e saída. O nível de ligação entre registros é aprimorado considerando-se a hierarquia dos objetos de teste em página, já que assim as similaridades são detectadas, preferencialmente, entre objetos de teste que estão no mesmo *layout* ou no mesmo módulo. A Figura 5.7 mostra as ligações de similaridades obtidas entre a entrada e a saída de uma mesma página.

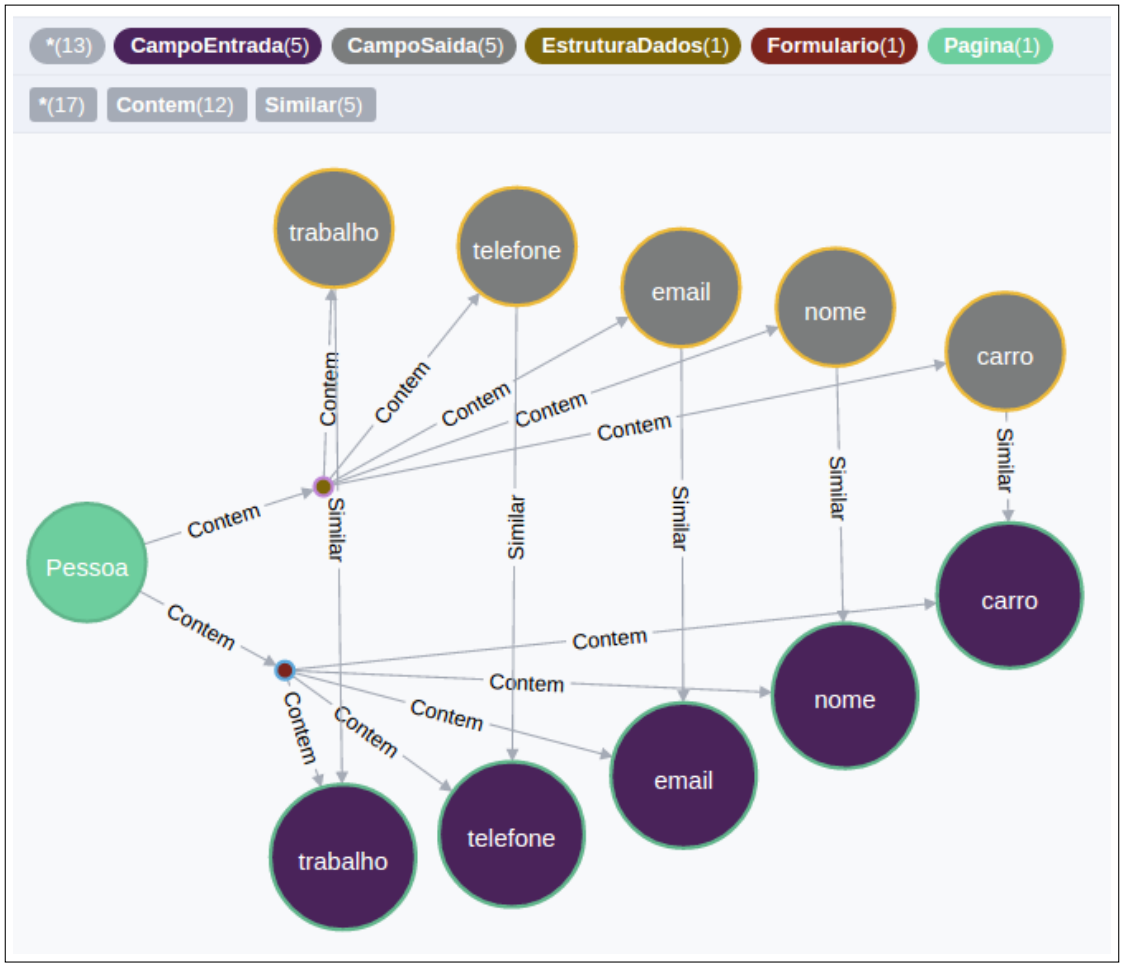

Figura 5.5: Estrutura em grafo de formulário extraída da AUT.

Fontes de dados também são obtidas com base na comparação de valores. O exemplo, na Figura 5.8, mostra um campo do tipo ‘comboBox‘, o qual contém sugestões para que o usuário possa selecionar um dos campos predefinidos. Nesse caso, é possível observar que os campos disponibilizados contêm os mesmos valores de uma fonte de dados em outro ponto do sistema. Com base nessas similaridades, é possível identificar fontes de dados mesmo que elas contenham um nome diferente.

5.5 Resultados de geração de casos de teste automatizados e execução

Após todas as ligações serem identificadas dentro do sistema, é necessário definir as ações que serão tomadas quando a execução for realizada, como por exemplo: cliques, preenchimentos ,etc. Os casos de teste de execução são obtidos por meio da junção das ligações já realizadas e das propriedades internas do nó. O tipo do objeto de teste (campo de texto, botão, *link*) determina qual ação será aplicada em tempo de automação. A Figura 5.9 demonstra o fluxo de execução criado a partir do SIATES.

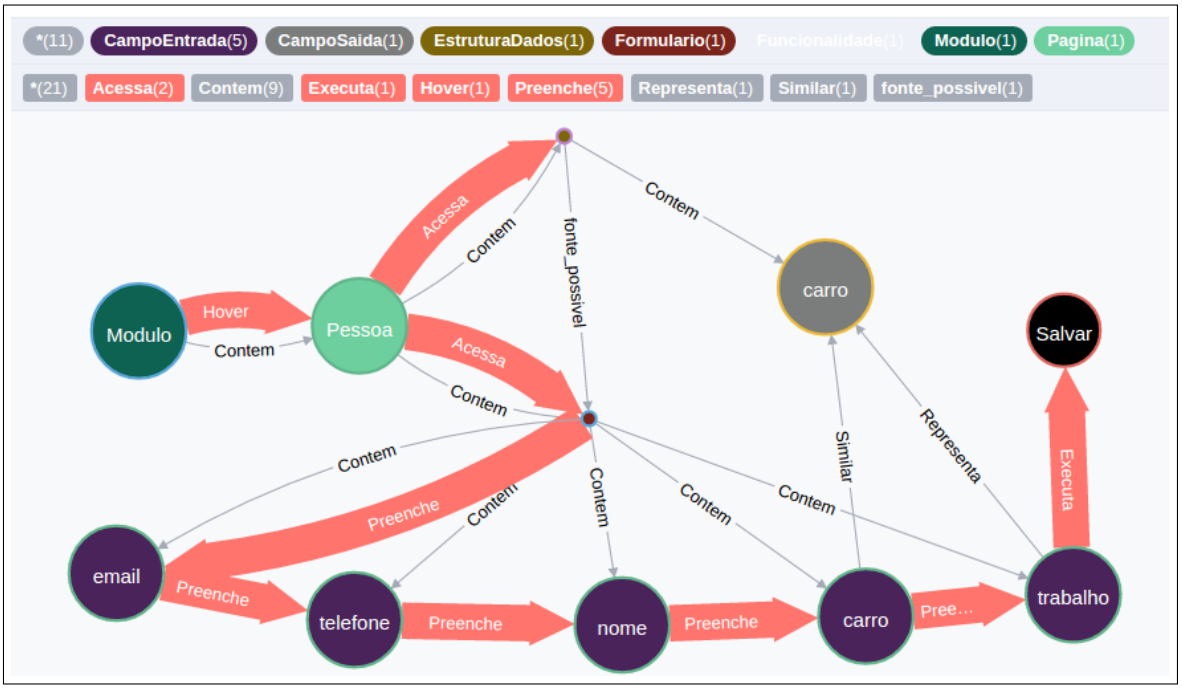

Figura 5.6: Estrutura de execução em grafo.

5.6 Resultados de classificação de mensagens de sistema

O experimento para o reconhecimento de mensagens de sistema foi realizado com 1.283 mensagens de três sistemas distintos, em que 914 representam mensagens de erro e 370 representam mensagens de sucesso. Essas mensagens foram previamente classificadas e separadas para a criação do modelo de aprendizagem.

Para confirmar a eficácia da classificação de mensagens com Maximização de Entropia, realizou-se um experimento usando a mesma base de dados com a ferramenta OpenNLP. Na realização dos testes do classificador de mensagens de status, o número de iterações ideais foi definido de acordo com a capacidade de melhoria do modelo. A Tabela 5.10 demonstra o número de iterações usado em cada teste correlacionada com a precisão de classificação. A precisão representa a quantidade de registros classificados de forma correta, de modo que

quanto mais próximo de 1, melhor.

Tabela 5.7: Comparativo de número de iterações e precisão na classificação de mensagens de sistema.

Iterações	precisão de classificação
50	0.972
100	0.975
500	0.985
592	0.987

O número de iterações ideais foi definido de acordo com a capacidade da aplicação de evoluir. Ao se realizar o teste com 1.000 iterações, observou-se que a precisão máxima é alcançada na iteração 592, de modo que o valor não aumenta independentemente do número a mais de iterações. Entre as mensagens usadas, extraíram-se 866 tokens de evento, com 324 predicados e duas classes de saída, sucesso e erro.

5.7 Execução de testes

Este experimento irá medir a eficiência da execução dos testes em relação a testadores manuais, considerando qualidade, identificação de erros e velocidade de execução. Para que isso ocorra a metodologia de teste de caixa preta é usada , este método busca testar o programa da mesma maneira que um usuário final testaria. Aqui, avalia-se o comportamento do software ao invés de sua implementação, de modo que o testador analisa as circunstâncias em que o programa não se comporta como deveria se comportar [Nidhra and Dondeti 2012].

O experimento de execução de testes realizados no Sistema Integrado de Planejamento e Orçamento (SIOP), que é o sistema responsável por auxiliar a realização das operações orçamentárias do Ministério do Planejamento e Secretaria de Orçamento Federal.

Tabela 5.8: Tipos de elementos de busca.

Tipo do objeto de teste	Descrição
Botões, *links* e objetos de teste clicáveis	Para esses objetos, o texto contido no objeto de teste e recomendado, o campo '*title*' pode ser usado como alternativa se o botão ou link não contiverem texto.
Elementos identificados por *label* (*checkboxes*, *radio-buttons* e texto, campos de texto, *selects*, etc.)	São campos que não guardam o valor do seu identificador em si mesmo, e dependem de um objeto de teste externo para identifica-los.

5.7.1 Detecção de erros

Esta fase do estudo analisa as discrepâncias entre o teste automatizado e o teste manual. Essa avaliação foi realizada com um grupo experimental composto de participantes de diferentes níveis educacionais e com experiência em realização de teste, conforme detalhado na Tabela 5.12 e 5.13.

Tabela 5.9: Descrição de experiência e formação dos participantes.

Participante	Grupo	Formação	Teste	AUT
1	1	Ensino médio	0	0
2	1	Ensino médio	0	0
3	1	Ensino médio	0	0
4	2	Ensino superior	3	0
5	2	Ensino superior	3	0
6	2	Ensino superior	2	0
7	3	Ensino superior	4	4
8	3	Ensino superior	3	3

O grupo de estudo foi composto por oito pessoas com níveis de escolaridade distribuídos entre ensino médio e superior. Os participantes foram divididos em três grupos de acordo com o nível de conhecimento em teste de *software*, conforme apresentado na Tabela 5.14. Os participantes tiveram de executar dez casos de testes, divididos em três avaliações e detalhados de forma que pudessem ser executados facilmente por uma pessoa, seguindo um modelo similar ao usado na ferramenta de automação.

Tabela 5.10: Descrição dos grupos.

Grupo	Descrição
Grupo 1	Participantes que nunca trabalharam com o testes
Grupo 2	Participantes que trabalham com o testes mas não conhecem o SIOP
Grupo 3	Participantes que trabalham realizando testes no SIOP

Com o objetivo de nivelar os conhecimentos dos participantes, ofereceu-se um treinamento aos envolvidos. Nesse treinamento, apresentaram-se os conceitos de teste de *software*, casos de teste e uma apresentação da aplicação sobre testes usada na avaliação. Para o estudo, foram criados casos de testes manuais com diferentes níveis de dificuldade.

Os casos de teste criados foram baseados em um fluxo principal para cada funcionalidade. Os participantes fizeram três execuções simulando situações de teste com graus variados de dificuldade. A primeira avaliação foi considerada mais fácil, já que somente exigia interação com uma funcionalidade e nenhum registro de dados no sistema, somente busca. A segunda

avaliação, com nível de dificuldade médio, continha somente uma operação de edição em um único módulo. A terceira avaliação foi considerada a mais complexa, pois o participante teve de entrar com dados no sistema, editar e deletar para interagir com múltiplos módulos. A seguir, na Figura 5.10, está detalhado um exemplo de um dos roteiros.

RESTAURAR AÇÃO	Operação realizada?	Status
Acessar URL: http://testes/siop/?pp=siop&rvn=1	Sim	Aprovado
Clicar na opção: Acesso SIOP	Sim	Aprovado
Inserir no campo "CPF:" o valor: 12116114187	Sim	Aprovado
Inserir no campo Senha: o valor: 123	Sim	Aprovado
Selecionar perfil: SOF	Sim	Aprovado
Clique no botão: Entrar	Sim	Aprovado
Selecionar exercício: 2016	Sim	Aprovado
Clique no botão: Selecionar	Sim	Aprovado
Deixar o cursor do mouse sobre o menu: LOA	Sim	Aprovado
Clicar no submenu: Ação	Sim	Aprovado
No campo Ação Inserir o valor: Avaliação 2 <<nome do testador>>	Sim	Aprovado
Clicar no botão "Procurar"	Sim	Aprovado
A ação deverá aparecer com o símbolo	Sim	Aprovado
Clique no link da ação que foi pesquisada	Sim	Aprovado
Clique no botão: Restaurar	Sim	Aprovado
Mensagem de confirmação : Ação restaurada com sucesso.	Sim	Aprovado
Clique no botão: Validar	Sim	Aprovado
Mensagem de confirmação : Ação validada com sucesso.	Sim	Aprovado
Após verificar o resultado esperado, clique no botão: Sair	Sim	Aprovado

Figura 5.7: Exemplo de roteiro de teste utilizado.

5.7.1.1 Avaliação 1

A avaliação é composta por três testes manuais, com os seus respectivos roteiros, na modalidade de testes de caixa preta.

Cada um dos testes foi elaborado com uma série de asserções e o candidato deveria marcar 'Sim' ou 'Não'. A marcação 'Sim' corresponderia à operação realizada com sucesso. Se o candidato marcasse a opção 'Não', aquela afirmativa seria reprovada e o candidato deveria descrever o motivo da negativa, ou seja, relatar qual foi o problema encontrado. Caso o participante achasse algum erro, também seria necessário que ele indicasse, no campo específico, o problema encontrado. Informações sobre os testes individuais estão detalhados na Tabela 5.14.

5.7.1.2 Avaliação 2 :

A avaliação 2 é composta por quatro testes com maior grau de dificuldade. Por não se tratar apenas de validações e pesquisa, nesta avaliação também é necessário realizar cadastro, o que exige atenção no que se refere ao uso dos valores corretos, de forma que o sistema não retorne uma mensagem de erro. A Tabela 5.15 mostra os detalhes.

Tabela 5.11: Resultados da avaliação de execução de testes manuais do grupo 1.

Teste	Descrição
Teste 1	O teste 1 não há erros, de forma que o usuário deverá somente validar a saída de sistema com o valor informado no *script*.
Teste 2	No Teste 2 há somente um defeito, a mensagem de erro apresentada no sistema está incorreta, porém somente duas palavras estão diferentes, o que requer atenção do testador
Teste 3	No teste 3 há dois erros, um dos valores pesquisados em tela não é o mesmo que está no roteiro, e a mensagem de erro retornada é diferente da esperada

Tabela 5.12: Resultados da avaliação de execução de testes manuais do grupo 2.

Avaliação	Pessoa	Erros de lógica	Erros encontrados	Erros de layout	Erros de layout encontrados
1	7	3	0	3	1
	8	3	0	3	1
2	7	2	0	3	1
	8	2	1	3	2
3	7	2	0	3	2
	8	2	2	3	3

5.7.1.3 Avaliação 3

A avaliação 3 é composta por três testes que apresentam maior dificuldade, já que esses testes dependem de testes executados em avaliações anteriores. Nessa avaliação, também é necessário que o participante entre no sistema com diferentes perfis para realizar as diferentes tarefas, conforme descrito na Tabela 5.17

Tabela 5.13: Resultados da avaliação de execução de testes manuais do grupo 3.

Teste	Descrição
Teste 1	O roteiro referenciava um campo que não existe em tela, o participante deve marcar o erro, e descreve-lo.
Teste 2	O Teste 2 não havia erros, de forma que o usário deve somente seguir o fluxo com atenção
Teste 3	O teste 3 também não contém nenhum erro
Teste 4	O roteiro fazia referencia a um botão que não existe em tela

Durante as simulações, foram inseridos 9 erros propositais dentro do sistema, a fim de verificar a habilidade dos participantes para identificar tais erros. Os erros foram compostos de registros inseridos com parâmetros diferentes dos esperados. Aos erros propositais, também foram adicionados outros 7 erros de layout, totalizando 16 erros, distribuídos em 3 avaliações. Para a análise dos resultados, foram considerados, na identificação de erros do sistema, os erros propositais, a inserção de dados e o tempo de execução.

Analisando os resultados obtidos no primeiro grupo, observou-se que houve maior difi-

culdade para concluir os testes. Um dos participantes identificou um erro de dados inserido no SIOP. Todos os outros erros não foram identificados. Em relação ao tempo de execução dos testes, na avaliação 1 (nível dificuldade menor), o tempo oscilou entre 1 minuto e 47 segundos a 8 minutos e 22 segundos. Na avaliação 2 (nível de dificuldade médio), o tempo máximo era 20 minutos, de modo que alguns participantes não conseguiram terminar o teste. Na avaliação 3 (nível de dificuldade maior), nenhum participante foi capaz de terminar dentro do espaço de tempo definido. Os resultados obtidos estão disponíveis na Tabela 5.17. O **'X'** representa um teste que o testador não conseguiu executar.

Tabela 5.14: Resultados da avaliação de execução de testes manuais do grupo 1 aplicado a ambiente com módulos interconectados.

Avaliação	**Pessoa**	**Erros de lógica**	**Erros encontrados**	**Erros de layout**	**Erros de layout encontrados**
1	1	3	0	3	0
	2	3	0	3	1
	3	3	0	3	0
2	1	2	0	3	0
	2	2	0	3	0
	3	2	0	3	0
3	1	2	**X**	3	**X**
	2	**X**	3	3	**X**
	3	2	1	3	**X**

Os participantes do grupo 2 encontraram 2 erros e os demais participantes encontraram somente 1 erro. Na segunda avaliação, dos 11 erros que deveriam ser encontrados por cada participante, foram encontrados somente 6 erros. Outros erros não foram encontrados. A respeito do tempo de execução, os participantes utilizaram entre 2 minutos e 14 segundos a 50 segundos avaliação 1; 6 minutos e 48 segundos a 20 minutos na avaliação 2; e nenhum dos participantes foi capaz de realizar a avaliação 3, conforme apresentado na Tabela 5.18, o 'X' representa um teste que o testador não conseguiu finalizar.

Tabela 5.15: Resultados da avaliação de execução de testes manuais do grupo 2 aplicado a ambiente com módulos interconectados.

Avaliação	Pessoa	Erros de lógica	Erros encontrados	Erros de layout	Erros de layout encontrados
1	4	0	2	3	0
	5	3	0	3	1
	6	3	0	3	1
2	4	2	1	3	0
	5	2	0	3	0
	6	2	0	3	0
3	4	2	**X**	3	**X**
	5	2	**X**	3	**X**
	6	2	0	3	**X**

Aos participantes do grupo 3, foi disponibilizado o menor tempo de execução e somente um dos participantes terminou essa avaliação de nível de dificuldade maior. Com relação aos erros da avaliação 1, somente um erro foi identificado. Na avaliação 2, foram identificados todos os 4 erros. Na avaliação 3, somente 3 erros não foram identificados. O resultado está disponível na Tabela 5.19.

Tabela 5.16: Resultados da avaliação de execução de testes manuais do grupo 3 aplicado a ambiente com módulos interconectados.

Avaliação	Pessoa	Erros de lógica	Erros encontrados	Erros de layout	Erros de layout encontrados
1	7	3	0	3	1
	8	3	0	3	1
2	7	2	0	3	1
	8	2	1	3	2
3	7	2	0	3	2
	8	2	2	3	3

Com a finalização dos testes, pode-se concluir que a qualidade geral dos testes foi menor do que o esperado nos grupos 2 e 3. Entre os fatores que possivelmente influenciaram na execução do teste incluem-se duração dos testes, fadiga, pressão por ser avaliado e tempo de execução dos testes. Os testes automatizados tiveram sucesso total para os *bugs* encontrados anteriormente, porém não identificaram padrões de layout errados e erros não definidos.

5.7.2 Legibilidade do caso de teste

A legibilidade do caso de teste afeta diretamente o tempo de manutenção dos testes. Com o uso do SIATES, pode-se observar que os objetos de teste são em geral mais identificáveis, por exemplo, enquanto o Selenium usa o localizador 'form:tblConsulta:0:CpfText', o SiopFit usa somente o localizador 'CPF' para referenciar o mesmo campo, identificando o campo por uma propriedade visível ao invés de uma propriedade interna.

O *framework* Rational Funcional Tester não usa propriedades internas para a localização de objetos. A ferramenta usa uma implementação interna com mapas de objetos baseando-se em múltiplas propriedades. Apesar de usar um recurso externo para a localização dos objetos de teste e os valores ainda são referenciados diretamente pelo código. Detalhes do caso de teste são apresentados na Figura 5.11.

```
1  // Search
2  link_formMenuJ_id2270J_id2330R().click();
3  text_nome().click(atPoint(141, 16));
4  browser_htmlBrowser(IntegradoD(),
5  DEFAULT_FLAGS).inputChars("Test automation");
6  list_formComboUnidadeFiltro().click();
7  list_formComboUnidadeFiltro().click();
8  list_formComboPerfilFiltro().click();
9  list_formComboPerfilFiltro().click();
10 text_cpf().click(atPoint(146, 11));
11 browser_htmlBrowser(
12 document_siopSistemaIntegradoD(),
13 DEFAULT_FLAGS).inputKeys("123123");
14 button_pesquisarsubmit().click();
15 html_formTblConsulta0DetalhePe().click(atPoint(50, 166));
16
17 // Edit
18 text_nome2().click(atPoint(129, 13));
19 browser_htmlBrowser(document_siopSistemaIntegradoD(), DEFAULT_FLAGS).inputChars("");
20 text_telefone().dragToScreenPoint(atPoint(100, 9), label_telefone().getScreenPoint(atPoint(2, 3)));
21 browser_htmlBrowser(document_siopSistemaIntegradoD(), DEFAULT_FLAGS).inputChars("123123");
22 button_salvarsubmit().click();
```

Figura 5.8: Exemplo de *script* para o *Rational Funcional Tester.*

O *script* faz a busca e a edição dentro de um registro em um sistema. A linha 2 clica um objeto de menu para navegar até a página de teste. A linha 2 clica um campo de texto. A linha 3 insere o texto *'Test automation'* no navegador. Entre as linhas 6 e 9, duas caixas de seleção são selecionadas; um novo texto é inserido no sistema; e, por fim, o botão de pesquisa é selecionado. A partir da linha 18, o processo é semelhante: clicando um registro existente e modificando o valor de telefone antes de executar o botão de salvar.

5.8 Comparação do SIATES com outras soluções similares

Existem outros trabalhos que buscam automatizar testes de *software*, em sua totalidade ou em partes, conforme apresentado no SIATES, usando diversas técnicas como heurística, *Finite State Machine* (FMS) e *Search Based Software Testing* (SBST). A seguir, são apresentadas as soluções aqui utilizadas com seus respectivos estudos.

5.8.1 *Automating test automation*

A solução apresentada pela IBM [Yandrapally et al. 2014] é aplicada também em sistemas *web*. Ela é considerada uma linguagem estilizada de alto nível devido ao fato de os casos de testes serem escritos em um padrão predeterminado. No entanto, não existem métodos para a criação de *scripts* automaticamente, somente a tradução dos de alto nível.

5.8.2 *A partical Approach for Automated Test Case Generaton using Statecharts*

A partir de um estudo realizado em Campinas, no Instituto Nacional de Pesquisa Espacial [Nebut et al. 2006], foi desenvolvida uma abordagem para a criação de casos de testes automático usando diagramas como base de criação dos casos de teste.

Apesar de o método da criação dos casos de teste ser simplificado e semi-automatizado, é necessário que o mapeamento das funcionalidades do sistema seja feita pelo testador. A abordagem apresentada automatiza boa parte do processo e consegue simplificar casos de testes em comparação com outras ferramentas.

5.8.3 *A Systematic Review of the Application and Empirical Investigation of Search-based TestCase Generation*

A solução proposta por este autor usa um método intitulado SBST [Afzal et al. 2009]. Devido ao fato de ser baseada em heurística, essa abordagem demonstra vantagens no que se refere à redução de custos. Os testes são executados exaustivamente na AUT, de forma que a solução possa aprender a testar o *software*.

A abordagem demonstrou sucesso em testar métodos isolados de uma AUT, também conhecidos como testes unitários. Tais testes têm baixa complexidade em comparação com testes em páginas de internet, pois considera menos fatores. Em contrapartida, testes orientados a ambientes web devem considerar tempo de carregar, fluxo de teste, renderização dos obejtos de teste, busca de objetos de teste e complexidade hierárquica da interface.

5.8.4 Comparação das ferramentas

A avaliação das ferramentas foi feita por meio da comparação dos parâmetros que focam na autonomia da ferramenta para a criação de casos de testes e facilidade na criação e na manutenção por testadores, apresentados na Tabela 5.20. Essas ferramentas são:

- Avaliação dos caminhos de teste: Esta característica mede a capacidade da ferramenta de obter os fluxos de execução de testes dentro da aplicação, permitindo a identifi-

cação de múltiplas rotas alternativas para testar a mesma funcionalidade, de forma a aumentar a qualidade de teste além dos caminhos de teste comum.

- Uso de uma linguagem de alto nível: Esta propriedade identifica a capacidade da ferramenta de usar uma linguagem simplificada de alto nível para manter os casos de teste. O que contribui para a facilidade de uso, economia de custos e velocidade na manutenção dos casos de teste.

- Criação automática de valores de teste: A criação automática de valores de teste dispensa o testador de ter que fornecer via ele mesmo, ou determinar uma fonte de onde os dados serão retirados.

- Uso em aplicações *web*: Ferramentas que contenham essa característica são capazes de lidar com as complexidades associadas a automação de testes para *web*.

- Avaliação de mensagem de status: Essa propriedade auxilia na independência, fazendo com que não seja necessária intervenção humana para avaliar se a execução do caso de teste acabou em sucesso ou erro.

Tabela 5.17: Resultados da comparação entre as ferramentas já citadas e o SIATES.

	Criação de Caminhos de teste	**Linguagem de alto nível**	**Parametrização da linguagem**	**Uso em aplicações web**	**Avaliação de mensagens de status**
[Thummalapenta et al. 2012]	Não	Sim	Não	Sim	Não
[Nebut et al. 2006]	Sim	Sim	Não	Sim	Não
[Afzal et al. 2009]	Sim	Não	Não	Não	Não
SIATES	Sim	Sim	Sim	Sim	Sim

A Tabela 5.17 compara as funcionalidades e aplicação entre o SIATES e outras ferramentas apresentadas anteriormente. Apesar das ferramentas concorrentes apresentarem algumas das vantagens, todas as vantagens não estão presentes na mesma ferramenta como no SIATES.

Capítulo 6

Conclusão

Nesta dissertação de mestrado, realizou-se o estudo de um modelo de Aprendizado de Máquina com Algoritmos Genéticos e Bancos de dados em grafos para automatização de testes em aplicações *web*. Nessa perspectiva, realizou-se um estudo sobre os problemas e soluções para implementação de automação de testes em ambientes de desenvolvimento de *software*, conforme apresentado no Capítulo 3.

Neste contexto foi possível confirmar que a automatização de testes de software utilizando a proposta de arquitetura apresentada no Capítulo 4 tal foi viabilizada conforme os resultados apresentados no Capítulo 5.

Com o uso de uma interface gráfica na camada de apresentação proposta, foi possível comprovar uma melhor performance no decorrer dos processos de teste, minimizando e facilitando a interação do testador, através da substituição da análise de código fonte bruto pela interação com uma interface gráfica mais amigável.

Através do levantamento bibliográfico levantado apresentado no Capítulo 2 foi possível comprovar que o uso de processamento de linguagem natural pode ser aplicado eficientemente no processo de classificação de texto. Na solução apresentada a técnica foi utilizada com sucesso para classificação de mensagens de *status*, permitindo identificar automaticamente os resultados de uma operação em um sistema *web* com uma precisão de 98.7%.

Na camada de processamento a utilização de um perceptron multicamada de uma rede neural artificial mostrou-se útil na classificação de objetos de teste com uma precisão de 98.89% de acerto. Através dessa classificação em conjunto com técnicas de extração de conteúdo e automatização de testes, foi viabilizada a interpretação automática de páginas *web*, tornando a análise manual desnecessária.

Outro ponto importante no processo de automação é o mapeamento e interpretação da estrutura da AUT, processo que é por vezes feito manualmente, onde sua complexidade de execução é exponencialmente relacionada a complexidade e tamanho do sistema. Foi possível concluir que o uso de banco de dados em grafo mostrou-se útil ao possibilitar a identificação de forma visual das ligações entre os diferentes pontos do sistema, viabilizando

a identificação estrutural e navegacional da aplicação sobre teste.

A utilização de AG para geração dos valores para os testes teve como embasamento o trabalho realizado por [Michael et al. 1997], conforme apresentado no Capítulo 5. Nesse sentido foi possível concluir que a utilização de uma abordagem que internaliza o processo de validação da qualidade dos dados possibilita a independência dessa etapa alcançando de 77 a 100 % de acertos para as gerações propostas nos experimentos, diferentemente da proposta que foi utilizada como referência, onde o autor propôs o uso de AG de forma dependente da AUT.

Apesar do progresso obtido com os métodos apresentados, existem processos que atualmente no SIATES não são passíveis a serem automatizados. Regras de negócio da aplicação devem ser validadas por um testador, de forma que os casos de teste devem ser avaliados após sua criação.

Os pontos já apresentados convergem para a minimização do envolvimento humano na automação de testes, sobretudo nos processos maçantes e repetitivos. Assim, os recursos podem ser deslocados para a parte gerencial do processo, como seleção dos casos de teste, avaliação dos resultados.

6.1 Trabalhos futuros

Por fim, recomenda-se, para este trabalho, o aumento performático da ferramenta bem como o aumento da sua flexibilidade, a fim de abranger o teste de outras aplicações que não sejam somente de cadastro e pesquisa. Ademais, os seguintes itens devem ser implementados no futuro:

- Aprimoramento da geração de dados de teste, para que seja possível identificar padrões mais complexos, por exemplo, dados que usam fórmulas matemáticas para serem criados como CPFs e números de cartão de crédito.
- Expandir o método de navegação e o reconhecimento de menus e páginas, com o objetivo da ferramenta ser capaz de navegar por meio de outras estruturas além de menus.
- Identificar as mudanças de *layout* de uma página, afim de verificar se a aplicação trouxe uma outra página mesmo que não tenha sido por um objeto de teste de menu. Esse padrão é encontrado em aplicações *one page application*.
- Identificar ações ocorridas no sistema além dos métodos básicos de *Create, Read, Update, Delete* (CRUD), sendo feito por meio da identificação de verbos nos objetos de teste que realizam ações por meio de *POS Tagging*.
- Extração de dados por outros meios que não sejam tabelas HTML, como listas e outras estruturas.

6.2 Publicações do autor

- Vitor, F et all *'TDD aplicado à sistemas web dinâmicos'* Conferencia Ibero Americana WWW/Internet (2016)

REFERÊNCIAS BIBLIOGRÁFICAS

[Abelson et al. 1996] Abelson, H., Sussman, G. J., and Sussman, J. (1996). *Structure and interpretation of computer programs*. Justin Kelly.

[Afzal et al. 2009] Afzal, W., Torkar, R., and Feldt, R. (2009). A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976.

[Al-Arashi et al. 2014] Al-Arashi, W. H., Ibrahim, H., and Suandi, S. A. (2014). Optimizing principal component analysis performance for face recognition using genetic algorithm. *Neurocomputing*, 128:415–420.

[Anand et al. 2013] Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., Harman, M., Harrold, M. J., Mcminn, P., et al. (2013). An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001.

[Anton 2005] Anton, T. (2005). Xpath-wrapper induction by generalizing tree traversal patterns. In *Lernen, Wissensentdeckung und Adaptivitt (LWA) 2005, GI Workshops, Saarbrcken*, pages 126–133.

[Ayala and dos Santos Coelho 2012] Ayala, H. V. H. and dos Santos Coelho, L. (2012). Tuning of pid controller based on a multiobjective genetic algorithm applied to a robotic manipulator. *Expert Systems with Applications*, 39(10):8968–8974.

[Barceló Baeza 2013] Barceló Baeza, P. (2013). Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 175–188. ACM.

[Bebis and Georgiopoulos 1994] Bebis, G. and Georgiopoulos, M. (1994). Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31.

[Bhatt et al. 2013] Bhatt, H. S., Bharadwaj, S., Singh, R., and Vatsa, M. (2013). Recognizing surgically altered face images using multiobjective evolutionary algorithm. *IEEE Transactions on Information Forensics and Security*, 8(1):89–100.

[Bishop 1995] Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

[Bodenhofer 2003] Bodenhofer, U. (2003). Genetic algorithms: theory and applications.

[Bolin and Miller 2005] Bolin, M. and Miller, R. C. (2005). Naming page elements in end-user web automation. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM.

[Booker et al. 1989] Booker, L. B., Goldberg, D. E., and Holland, J. H. (1989). Classifier systems and genetic algorithms. *Artificial intelligence*, 40(1):235–282.

[Chowdhury 2003] Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, 37(1):51–89.

[Coley 1999] Coley, D. A. (1999). *An introduction to genetic algorithms for scientists and engineers*. World scientific.

[Crescenzi et al. 2001] Crescenzi, V., Mecca, G., Merialdo, P., et al. (2001). Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, volume 1, pages 109–118.

[Da Silva and Menezes 2005] Da Silva, E. L. and Menezes, E. M. (2005). Metodologia da pesquisa e elaboração de dissertação. *UFSC, Florianópolis, 4a. edição*, 123.

[Dalvi et al. 2012] Dalvi, B. B., Cohen, W. W., and Callan, J. (2012). Websets: Extracting sets of entities from the web using unsupervised information extraction. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 243–252. ACM.

[Darwin 1872] Darwin, C. (1872). *The origin of species*. Lulu. com.

[Deb et al. 2002] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197.

[Douce et al. 2005] Douce, C., Livingstone, D., and Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4.

[Dustin 2002] Dustin, E. (2002). *Effective Software Testing: 50 Ways to Improve Your Software Testing*. Addison-Wesley Longman Publishing Co., Inc.

[Girgis 2005] Girgis, M. R. (2005). Automatic test data generation for data flow testing using a genetic algorithm. *J. UCS*, 11(6):898–915.

[Glover et al. 2002] Glover, E. J., Tsioutsiouliklis, K., Lawrence, S., Pennock, D. M., and Flake, G. W. (2002). Using web structure for classifying and describing web pages. In *Proceedings of the 11th international conference on World Wide Web*, pages 562–569. ACM.

[Gottlob et al. 2005] Gottlob, G., Koch, C., and Pichler, R. (2005). Efficient algorithms for processing xpath queries. *ACM Transactions on Database Systems (TODS)*, 30(2):444–491.

[Haykin and Network 2004] Haykin, S. and Network, N. (2004). A comprehensive foundation. *Neural Networks*, 2(2004).

[Hepner 1990] Hepner, G. F. (1990). Artificial neural network classification using a minimal training set. comparison to conventional supervised classification. *Photogrammetric Engineering and Remote Sensing*, 56(4):469–473.

[Kifetew et al. 2013] Kifetew, F. M., Panichella, A., De Lucia, A., Oliveto, R., and Tonella, P. (2013). Orthogonal exploration of the search space in evolutionary test case generation. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, pages 257–267. ACM.

[Kimoto et al. 1990] Kimoto, T., Asakawa, K., Yoda, M., and Takeoka, M. (1990). Stock market prediction system with modular neural networks. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 1–6. IEEE.

[Korel 1990] Korel, B. (1990). Automated software test data generation. *IEEE Transactions on software engineering*, 16(8):870–879.

[Korel 1996] Korel, B. (1996). Automated test data generation for programs with procedures. In *ACM SIGSOFT Software Engineering Notes*, volume 21, pages 209–215. ACM.

[Little et al. 2007] Little, G., Lau, T. A., Cypher, A., Lin, J., Haber, E. M., and Kandogan, E. (2007). Koala: capture, share, automate, personalize business processes on the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 943–946. ACM.

[Little and Miller 2006] Little, G. and Miller, R. C. (2006). Translating keyword commands into executable code. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 135–144. ACM.

[Mahmud and Lau 2010] Mahmud, J. and Lau, T. (2010). Lowering the barriers to website testing with cotester. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 169–178. ACM.

[McCallum et al. 1998] McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.

[McMinn 2004] McMinn, P. (2004). Search-based software test data generation: A survey. *Software Testing Verification and Reliability*, 14(2):105–156.

[Medsker and Jain 2001] Medsker, L. and Jain, L. (2001). Recurrent neural networks. *Design and Applications*, 5.

[Memon et al. 2001] Memon, A. M., Pollack, M. E., and Soffa, M. L. (2001). Hierarchical gui test case generation using automated planning. *IEEE transactions on software engineering*, 27(2):144–155.

[Memon and Soffa 2003] Memon, A. M. and Soffa, M. L. (2003). Regression testing of guis. *ACM SIGSOFT Software Engineering Notes*, 28(5):118–127.

[Michael et al. 1997] Michael, C. C., McGraw, G. E., Schatz, M. A., and Walton, C. C. (1997). Genetic algorithms for dynamic test data generation. In *Automated Software Engineering, 1997. Proceedings., 12th IEEE International Conference*, pages 307–308. IEEE.

[Michalski et al. 2013] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.

[Mitchell 1998] Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

[Myers and Miller 1988] Myers, E. W. and Miller, W. (1988). Optimal alignments in linear space. *Computer applications in the biosciences: CABIOS*, 4(1):11–17.

[Nebut et al. 2006] Nebut, C., Fleurey, F., Le Traon, Y., and Jezequel, J.-M. (2006). Automatic test generation: A use case driven approach. *IEEE Transactions on Software Engineering*, 32(3):140–155.

[Neo4J 2016] Neo4J (2016). Neo4j. *disponível em : https://neo4j.com/ Acessado em : 2016-09-13*.

[Nidhra and Dondeti 2012] Nidhra, S. and Dondeti, J. (2012). Blackbox and whitebox testing techniques-a literature review. *International Journal of Embedded Systems and Applications (IJESA)*, 2(2):29–50.

[Pargas et al. 1999] Pargas, R. P., Harrold, M. J., and Peck, R. R. (1999). Test-data generation using genetic algorithms. *Software Testing Verification and Reliability*, 9(4):263–282.

[Parveen and Tilley 2010] Parveen, T. and Tilley, S. (2010). When to migrate software testing to the cloud? In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, pages 424–427. IEEE.

[Paulinas and Ušinskas 2015] Paulinas, M. and Ušinskas, A. (2015). A survey of genetic algorithms applications for image enhancement and segmentation. *Information Technology and control*, 36(3).

[Pressman 2005] Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

[Processing 1986] Processing, P. D. (1986). Explorations in the microstructure of cognition. vol. 1: Foundations. *Rumelhart et al*, page 318.

[Purnamasari et al. 2013] Purnamasari, D., Wicaksana, I. W. S., Harmanto, S., and Banowosari, L. Y. (2013). Html extraction algorithm based on property and data cell. In *IOP Conference Series: Materials Science and Engineering*, volume 46, page 012035. IOP Publishing.

[Rafi et al. 2012] Rafi, D. M., Moses, K. R. K., Petersen, K., and Mäntylä, M. V. (2012). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *Proceedings of the 7th International Workshop on Automation of Software Test*, pages 36–42. IEEE Press.

[Robinson 2015] Robinson, I. (2015). *Graph Databases*. O'Reilly.

[Sonka et al. 2014] Sonka, M., Hlavac, V., and Boyle, R. (2014). *Image processing, analysis, and machine vision*. Cengage Learning.

[Specht 1990] Specht, D. F. (1990). Probabilistic neural networks. *Neural networks*, 3(1):109–118.

[Srivastava and Kim 2009] Srivastava, P. R. and Kim, T.-h. (2009). Application of genetic algorithm in software testing. *International Journal of software Engineering and its Applications*, 3(4):87–96.

[Tengli et al. 2004] Tengli, A., Yang, Y., and Ma, N. L. (2004). Learning table extraction from examples. In *Proceedings of the 20th international conference on Computational Linguistics*, page 987. Association for Computational Linguistics.

[Tuncer and Yildirim 2012] Tuncer, A. and Yildirim, M. (2012). Dynamic path planning of mobile robots with improved genetic algorithm. *Computers & Electrical Engineering*, 38(6):1564–1572.

[Van Bruggen 2014] Van Bruggen, R. (2014). *Learning Neo4j*. Packt Publishing Ltd.

[Vidal et al. 2013] Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489.

[Widrow and Hoff 1960] Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. Technical report, STANFORD UNIV CA STANFORD ELECTRONICS LABS.

[Wong et al. 2009] Wong, W., Martinez, D., and Cavedon, L. (2009). Extraction of named entities from tables in gene mutation literature. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, pages 46–54. Association for Computational Linguistics.

[Wood 2012] Wood, P. T. (2012). Query languages for graph databases. *ACM SIGMOD Record*, 41(1):50–60.

[Xu et al. 2007] Xu, R., Wunsch II, D., and Frank, R. (2007). Inference of genetic regulatory networks with recurrent neural network models using particle swarm optimization. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):681–692.

[Yandrapally et al. 2014] Yandrapally, R., Thummalapenta, S., Sinha, S., and Chandra, S. (2014). Robust test automation using contextual clues. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 304–314. ACM.

[Zhu et al. 2005] Zhu, S., Ji, X., Xu, W., and Gong, Y. (2005). Multi-labelled classification using maximum entropy method. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 274–281. ACM.

Apêndice

Aqui é apresentado os anexos do trabalho, incluindo amostras de dados e código fonte usados no SIATES.

Anexo A : Classificação de mensagens de *status* de sistema

Aqui é exemplificado as amostras de mensagens de status de sistema, sendo elas mensagens de erro e mensagens de sucesso. Essas mensagens foram usadas para treinar um algoritmo de classificação, detalhado nos capítulos 4 e 5.

Anexo A.1 : Amostra de mensagens de erro

Erro ao salvar. Favor tente novamente.
Solicitação de Local já cadastrada.
Preenchimento Obrigatório.
Operação cancelada. Já existe registro cadastrado com essa descrição.
CNPJ Inválido.
CPF Inválido.
Falha ao enviar email, tente novamente realizar o cadastro.
A chave utilizada está expirada.
A chave informada está incorreta ou nehum usuário cadastrado.
Usuário já cadastrado.
Email preenchido incorretamente.
Número de Folhas informado incorreto.
É obrigatório informar campo.
CEP Inválido.

Anexo A.2 : Amostra de mensagens de sucesso

Utilização de Excesso aprovada com sucesso
Proposta enviada com sucesso!
Entidades capturadas com sucesso.
Encerrando a vigência...
Novo Registro inserido
Registro salvo com sucesso
Registro copiado
Registro(s) zerado(s) com sucesso
Registro salvo com sucesso
Senha de acesso atualizada com sucesso.
A senha atual será mantida.
Consulta copiada com sucesso
Configuração(ões) alterada(s) com sucesso!
Cópia realizada com sucesso!
Exclusão realizada com sucesso!
Registro recusado com sucesso!

Anexo B : Classificador de mensagens de texto

Aqui é exposto o código fonte do classificador de mensagens de sistema. O classificador foi escrito na linguagem de programação Java, usando o *framework* OpenNPL.

```
1
2  package reconhecimentoMensagem;
3
4  import java.io.File;
5  import java.io.FileInputStream;
6  import java.io.IOException;
7  import java.io.InputStream;
8  import java.io.PrintWriter;
9
10 import opennlp.tools.doccat.DoccatModel;
11 import opennlp.tools.doccat.DocumentCategorizerME;
12 import opennlp.tools.doccat.DocumentSampleStream;
13 import opennlp.tools.util.ObjectStream;
14 import opennlp.tools.util.PlainTextByLineStream;
15
16 /**
17 *
18 * @author Vitor Lopes
19 */
```

```
20 public class CategorizadorMensagens {
21
22         public static void main(String[] args) {
23                 CategorizadorMensagens categorizador=new
                       CategorizadorMensagens();
24                 categorizador.trainModel();
25         }
26
27         DoccatModel model;
28
29         public void trainModel() {
30                 InputStream dataIn = null;
31                 try {
32                         dataIn = new FileInputStream("messageTraining.txt
                               ");
33                         ObjectStream lineStream = new
                               PlainTextByLineStream(dataIn, "UTF-8");
34                         ObjectStream sampleStream = new
                               DocumentSampleStream(lineStream);
35
36
37                         model = DocumentCategorizerME.train("pt",
                               sampleStream,5,50);
38                 } catch (IOException e) {
39                         e.printStackTrace();
40                 } finally {
41                         if (dataIn != null) {
42                                 try {
43                                         dataIn.close();
44                                 } catch (IOException e) {
45                                         e.printStackTrace();
46                                 }
47                         }
48                 }
49         }
50
51         public String categorizarMensagem(String message) {
52                 trainModel();
53                 DocumentCategorizerME myCategorizer = new
                       DocumentCategorizerME(model);
54                 double[] outcomes = myCategorizer.categorize(message);
55                 String category = myCategorizer.getBestCategory(outcomes)
                       ;
56
57                 if (category.equalsIgnoreCase("1")) {
58                         return "Sucesso";
59                 } else {
60                         return "Erro";
61                 }
```

```
62          }
63
64
65  }
```

Anexo C : Classificação de objetos de teste

Este anexo detalha o classificador de objetos de teste, usando para diferenciar os objetos de entrada, saída e navegação dentro de uma página. O classificador foi escrito com a linguagem de programação Java e testado com a ferramenta Weka.

Anexo C.1 : Classe extratora de *features* de páginas HTML

O código a seguir foi usado para obter os objetos de teste que são usados para treino do classificador de objetos de teste.

```
1
2  package extratorFeature;
3
4  import java.io.BufferedReader;
5  import java.io.File;
6  import java.io.FileReader;
7  import org.jsoup.Jsoup;
8  import org.jsoup.nodes.Document;
9  import org.jsoup.nodes.Element;
10 import org.jsoup.select.Elements;
11
12 /**
13 * @author Vitor Lopes
14 */
15 public class ExtratorFeaturesHTML {
16
17         public static void main(String[] args) {
18                 try {
19                         ExtratorFeaturesHTML extrator = new
                            ExtratorFeaturesHTML();
20                         extrator.test();
21                 } catch (Exception e) {
22                         e.printStackTrace();
23                 }
24         }
25
26         public void test() {
27
28                 ExtratorFeaturesHTML extrator = new ExtratorFeaturesHTML
                    ();
```

```
29              String conteudoArquivo = extrator.obterTextoArquivo("
                    amostrasDados/amostras.txt");
30              Document doc = Jsoup.parse(conteudoArquivo);
31
32              String[] listaElementosAlvo = {"input", "a", "td", "label
                    ", "select", "button", "submit"};
33
34              for (String tipo : listaElementosAlvo) {
35
36                      Elements inputs = doc.getElementsByTag(tipo);
37
38                      for (Element elemento : inputs) {
39                              String tipoElemento = "menu";
40
41                              String elementoAlvo = elemento.tagName();
42                              String elementoAlvoPai = elemento.parent
                                    ().tagName();
43                              String elementoAlvoAvo = elemento.parent
                                    ().parent().tagName();
44                              String elementoAlvoBizaavo = elemento.
                                    parent().parent().parent().tagName();
45
46                              int quantidadeInputsIrmaos = elemento.
                                    parent().getElementsByTag("input").
                                    size();
47                              int quantidadeCelulasIrmaos = elemento.
                                    parent().getElementsByTag("tr").size()
                                    ;
48                              int quantidadeItemMenu = elemento.parent
                                    ().getElementsByTag("li").size();
49
50                              int quantidadeFilhos = elemento.children
                                    ().size();
51
52                              boolean contemFilhoTexto = true;
53                              if (elemento.text().trim().isEmpty()) {
54                                      contemFilhoTexto = false;
55                              }
56
57                              System.out.println(elementoAlvo
58                              + "," + elementoAlvoPai
59                              + "," + elementoAlvoAvo
60                              + "," + elementoAlvoBizaavo
61                              + "," + quantidadeInputsIrmaos
62                              + "," + quantidadeCelulasIrmaos
63                              + "," + quantidadeItemMenu
64                              + "," + quantidadeFilhos
65                              + "," + contemFilhoTexto + "," +
                                    tipoElemento);
```

```
66                          }
67
68                  }
69
70          }
71
72          public String obterTextoArquivo(String caminhoArquivo) {
73                  try {
74                          BufferedReader br = new BufferedReader(new
                                FileReader(caminhoArquivo));
75                          try {
76                                  StringBuilder sb = new StringBuilder();
77                                  String line = br.readLine();
78
79                                  while (line != null) {
80                                          sb.append(line);
81                                          sb.append(System.lineSeparator())
                                                ;
82                                          line = br.readLine();
83                                  }
84                                  String everything = sb.toString();
85                                  return everything;
86                          } finally {
87                                  br.close();
88                          }
89                  } catch (Exception e) {
90                          e.printStackTrace();
91                  }
92                  return null;
93          }
94  }
```

Anexo C.2 : Amostra de dados de *features* extraídas de formulários

Este anexo é uma amostra dos dados extraídos de formulários de cadastro de uma AUT. Esses formulários são usados como ponto de entrada de dados em um sistema.

tipo	alvo	pai	avô	bizavô	input irmão	célula irmão	menu irmão	quantidade filhos	contêm texto
formulário	input	fieldset	form	body	5	0	0	0	falso
formulário	input	div	form	body	2	0	0	0	falso
formulário	input	form	body	html	1	0	0	0	falso
formulário	input	form	body	html	5	0	0	0	falso
formulário	input	div	div	form	7	0	0	0	falso

Anexo C.3 : Amostra de dados de *features* extraídas de menus

Este anexo é uma amostra dos dados extraídos de menus de navegação de uma AUT. Esses menus são usados para navegação entre as diferentes partes do sistema.

tipo	alvo	pai	avô	bizavô	input irmão	célula irmão	menu irmão	quantidade filhos	contêm texto
menu	a	li	ul	li	5	0	0	0	verdadeiro
menu	a	li	ul	body	2	0	0	0	verdadeiro
menu	a	li	body	html	1	0	0	0	verdadeiro
menu	a	div	li	ul	0	0	4	0	verdadeiro
menu	a	div	li	ul	0	0	7	0	verdadeiro

Anexo C.4 :Amostra de dados de *features* extraídas de tabelas

Este anexo é uma amostra dos dados extraídos de elementos de saída de uma AUT. Esses elementos são responsáveis por exibir informações ao usuário.

tipo	alvo	pai	avô	bizavô	input irmão	célula irmão	menu irmão	quantidade filhos	contêm texto
tabela	td	tr	tbody	table	0	1	0	0	verdadeiro
tabela	td	tr	tbody	body	0	1	0	1	falso
tabela	td	tr	body	html	0	0	0	1	falso
tabela	td	tr	li	ul	1	1	0	0	verdadeiro
tabela	td	tr	li	ul	0	1	0	0	verdadeiro

Anexo D : Legenda das *features* da rede neural

Este anexo é demonstra todas as *features* das redes neurais, essa tabela inclui todas as variações entre as possibilidades de elementos HTML que são importantes para a classificação dos elementos.

Número	Feature
1	alvo=input
2	alvo=li
3	alvo=table
4	alvo=td
5	alvo=tr
6	alvo=label
7	alvo=select
8	alvo=button
9	alvo=a
10	alvo=b
11	pai=fieldset
12	pai=div
13	pai=form
14	pai=li
15	pai=label
16	pai=p
17	pai=span
18	pai=ul
19	pai=section
20	pai=tr
21	pai=tbody
22	pai=dd
23	pai=h4
24	pai=h3
25	pai=strong
26	pai=body
27	pai=nav
28	pai=article
29	pai=header
30	pai=b
31	pai=td
32	pai=thread
33	pai=tfoot
34	avo=form
35	avo=body
36	avo=div
37	avo=ul

Número	Feature
38	avo=label
39	avo=fieldset
40	avo=li
41	avo=p
42	avo=section
43	avo=span
44	avo=a
45	avo=tbody
46	avo=table
47	avo=h3
48	avo=dl
49	avo=nav
50	avo=article
51	avo=header
52	avo=html
53	avo=tr
54	avo=tfoot
55	avo=thread
56	bizavo=body
57	bizavo=div
58	bizavo=fieldset
59	bizavo=ul
60	bizavo=section
61	bizavo=label
62	bizavo=table
63	bizavo=li
64	bizavo=span
65	bizavo=root
66	bizavo=nav
67	bizavo=header
68	bizavo=tbody
69	inputirmao
70	celulairmao
71	menuirmao
72	contemfilho
73	contemtexto
74	quantidadefilhos