Universidade Federal do Piauí
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação

Um Estudo Prático sobre a Automação em Oráculos de Testes

Ronyérison Dantas Braga

Número de Ordem PPGCC: M001

Teresina-PI, 24 de Agosto de 2018

Ronyérison Dantas Braga

Um Estudo Prático sobre a Automação em Oráculos de Testes

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Universidade Federal do Piauí – UFPI

Centro de Ciências da Natureza

Programa de Pós-Graduação em Ciência da Computação

Orientador: Pedro de Alcântara dos Santos Neto

Teresina-PI

24 de Agosto de 2018

Ronyérison Dantas Braga
Um Estudo Prático sobre a Automação em Oráculos de Testes/ Ronyérison Dantas Braga. – Teresina-PI, 24 de Agosto de 2018-
102 p. : il. (algumas color.) ; 30 cm.

Orientador: Pedro de Alcântara dos Santos Neto

Dissertação (Mestrado) – Universidade Federal do Piauí – UFPI
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação, 24 de Agosto de 2018.

1. Oráculos de Testes. 2. Aprendizagem de Máquina. 3. Automação. 4. Método. 5. Ferramenta. I. Pedro de Alcântara dos Santos Neto. II. Universidade Federal do Piauí. III. Um Estudo Prático sobre a Automação em Oráculos de Testes

CDU 02:141:005.7

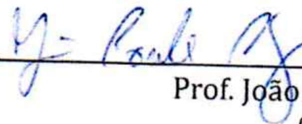
"Um Estudo Prático sobre a Automação em Oráculos de Testes"

RONYÉRISON DANTAS BRAGA

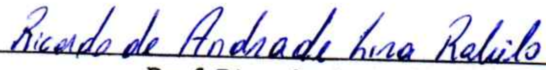
Dissertação apresentada ao Programa de Pós–Graduação em Ciência da Computação do Centro de Ciências da Natureza da Universidade Federal do Piauí, como parte integrante dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Aprovada por:

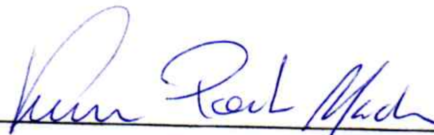
Prof. Pedro de Alcântara dos Santos Neto
(Presidente da Banca Examinadora)



Prof. João Paulo Pordeus Gomes
(Examinador Externo)



Prof. Ricardo de Andrade Lira Rabêlo
(Examinador Interno)



Prof.Vinicius Ponte Machado
(Examinador Externo)

Teresina, 24 de agosto de 2018

Aos meus pais Jonas de Araújo Braga e Fátima de Lurdes Dantas Braga,
por sempre estarem comigo em todos os momentos.

Agradecimentos

Agradeço a Deus. Pelas dificuldades que me tornaram mais forte para superá-las e assim conquistar meus objetivos.

Agradeço aos meus pais, Jonas de Araújo Braga e Fátima de Lurdes Dantas Braga, por todos os seus ensinamentos que me guiaram e me ajudaram em todos os momentos da minha caminhada. A eles devo tudo, são minha fonte de inspiração por sua dedicação com os filhos, sempre lutando para superar os desafios impostos pela vida. À minha irmã Sâmia, pelo companheirismo e apoio nessa árdua jornada.

Agradeço também à toda a minha família: avós, tios, tias, padrinhos, madrinhas, primos e primas. Pelo amor, carinho em todos os momentos, sempre foram a base para a minha formação como pessoa.

Agradeço ao meu orientador, Pedro de Alcântara dos Santos Neto, por todos os conselhos, pela paciência e ajuda nesse período. Sou muito grato por sua confiança e dedicação em todos esses anos de trabalho. Foram muitos ensinamentos, que não levo apenas para a vida acadêmica e sim para a vida. Sempre será uma referência como pessoa, professor e pesquisador.

Agradeço muito à minha namorada Flávia Gonçalves Teixeira, pelo amor, carinho e compreensão durante toda a minha caminhada. Sempre esteve comigo nos bons momentos e, principalmente, nos momentos mais difíceis. Sou infinitamente grato por tudo que tem feito por mim e por sua dedicação.

Aos meus amigos que fiz durante a graduação e mestrado. Foram bons momentos de aprendizado e diversão, o que tornou esse longo período muito mais prazeroso para chegar ao objetivo final.

Agradeço muito a todos que participaram dessa conquista. Foram muitos obstáculos superados com apoio das excelentes pessoas que me cercam, sozinho nunca conseguiria atingir os objetivos tão desejados, muito obrigado a TODOS.

“The people who are
crazy enough to think
they can change the world
are the ones who do.”
(Steve Jobs)

Resumo

A computação tem sido utilizada para apoio à solução de problemas nas mais diversas áreas de conhecimento, tais como medicina, biologia, matemática, mecânica, administração, economia, etc. Inúmeras aplicações têm sido desenvolvidas para auxiliar o ser humano na realização de tarefas que antes eram realizadas de forma manual, tais como, aplicações para identificação de anomalias em imagens médicas (medicina), aplicações para previsão de valores de ações no mercado financeiro (economia), aplicações para gestão patrimonial (administração), dentre outras. Entretanto, para garantir a confiabilidade das aplicações para solução de tais problemas é preciso que elas passem por uma etapa essencial no processo de desenvolvimento de software, o Teste de Software. O Teste de Software é uma das atividades que se concentra na gestão da qualidade de software, e pode ser definido como um conjunto de tarefas, planejadas e executadas sistematicamente com o propósito de descobrir erros cometidos durante a implementação dos softwares. Uma das tarefas mais complexas e custosas relacionada ao Teste de Software é o mecanismo popularmente conhecido como Oráculo de Testes. Dada uma determinada entrada para uma aplicação, a difícil tarefa de distinguir o comportamento correto para o comportamento potencialmente incorreto da aplicação é chamado de "Problema do Oráculo de Teste". Geralmente, essa atividade é realizada de forma manual por um desenvolvedor ou testador da aplicação, tornando-se um gargalo na realização de testes de software. Considerando esse contexto, é proposto neste trabalho uma abordagem, para automação do mecanismo de oráculo de testes. Essa proposta traz consigo a inovação na forma como a atividade é realizada e torna-se uma alternativa para as pesquisas já existentes dessa área. Este trabalho está relacionado a um projeto maior com a intenção de aplicar teste de regressão de forma contínua em softwares em ambiente de produção. A ideia é construir uma abordagem para automação do mecanismo de oráculo de teste que possa ser utilizada de forma não intrusiva nas aplicações em ambiente real. Após a realização de um estudo da área, foi proposta uma nova abordagem para automação do oráculo de teste. Foram realizadas quatro avaliações experimentais com o intuito de analisar a eficácia da abordagem na solução do problema do oráculo de teste. Os experimentos apresentaram resultados que sugerem que a ideia aqui apresentada é relevante para a solução do problema abordado.

Palavras-chaves: Oráculos de Testes, Aprendizagem de Máquina, Teste de Software, Automação, Aborgagem.

Abstract

The computing has been used to support problem-solving in a wide range of fields such as medicine, biology, mathematic, mechanic, administration, economy and others. Numerous applications have been developed to assist the human being in performing tasks that were previously performed manually, such as applications for identification of anomalies in medical images (medicine), applications for predicting stock values in the financial market (economy), applications for patrimonial management (administration), among others. However, to ensure the reliability of applications for solving such problems, it must pass by an essential step in the software development process, the Software Testing. The Software Testing is one of the activities that focus on software quality management and can be defined as a set of tasks, planned and executed systematically with the purpose of discovering errors made during software implementation. One of the most complex and costly tasks related to Software Testing is the mechanism popularly known as Test Oracle. Given a particular input to an application, the difficult task of distinguishing the correct behavior for the potentially incorrect behavior of the application is called "Test Oracle Problem". Generally, this task is performed manually by a developer or application tester, making it a bottleneck in performing of the software testing. Considering this context, it's proposed in this work an approach for automation of the test oracle mechanism. This proposal brings with it the innovation in the way the activity is performed and becomes an alternative for existing research in this area. This work is related to a larger project with the intention of applying regression testing continuously in software in the production environment. The idea is to build an approach to automation of the test oracle engine that can be used non-intrusively in real-world applications. After a study of the area, a new approach was proposed for automation of the test oracle. Four experimental evaluations were performed in order to analyze the effectiveness of the approach in solving the test oracle problem. The experiments presented results with suggest that the idea presented here is relevant to the solution of the problem addressed.

Keywords: Tests Oracles, Machine Learning, Software Testing, Automation, Approach.

Lista de ilustrações

Figura 1 – Número de publicações sobre Teste de Software Automático obtidas no *Scopus*. 4
Figura 2 – Exemplo de uma Curva ROC. 20
Figura 3 – Processo do MSE. 21
Figura 4 – Processo de Busca do MSE. 23
Figura 5 – Processo de Seleção de Estudos. 24
Figura 6 – Processo de extração de dados. 26
Figura 7 – Distribuição das publicações identificadas entre os anos de 2000 e 2016. 31
Figura 8 – Mapa Sistemático relacionando Contribuição e Tipo de Pesquisa. . . . 32
Figura 9 – Mapa Sistemático relacionando Contribuição e Técnica para automação. 36
Figura 10 – Mapa Sistemático para Restrições de Linguagem. 37
Figura 11 – Mapa Sistemático dos Tipos de Softwares avaliados. 41
Figura 12 – Mapa Sistemático que relaciona Técnica de Automação com Tipo de Teste de Software e Técnica de Automação com Tipos de Softwares. . . 43
Figura 13 – Abordagem proposta para automação do oráculo de teste. 51
Figura 14 – Exemplo ilustrando a etapa de captura de ações. 52
Figura 15 – Etapa de preparação de dados. 56
Figura 16 – Interface Gráfica da Ferramenta WEKA. 59
Figura 17 – Etapa de Aprendizagem de Máquina. 60
Figura 18 – Tecnologias utilizadas na implementação da ferramenta. 61
Figura 19 – Captura de tela da página de visualização de aplicações cadastradas. . 62
Figura 20 – Captura de tela com resultados de uma simulação. 63

Lista de tabelas

Tabela 1 – Exemplo de uma matriz de confusão para problemas binários. 18
Tabela 2 – Resultado da busca nas bases de dados. 24
Tabela 3 – Critérios para exclusão de trabalhos. 25
Tabela 4 – Locais de publicação com trabalhos aceitos no MSE. 31
Tabela 5 – Resultados da classificação dos trabalhos por Técnica de Automação. . 35
Tabela 6 – Resultados da classificação dos trabalhos em relação ao Tipo de Teste. 37
Tabela 7 – Mapa Sistemático para Tipos de Softwares. 38
Tabela 8 – Mapa Sistemático para a questão Métodos de Avaliação Empírica. . . . 39
Tabela 9 – Entradas e saídas do classificador. 58
Tabela 10 – Ganho de Informação dos atributos capturados. 69
Tabela 11 – Hiper-parâmetros de configuração dos algoritmos de AM. 70
Tabela 12 – Matriz de confusão da primeira avaliação. 73
Tabela 13 – Resultado do calculo das medidas da primeira avaliação experimental. . 73
Tabela 14 – Resultados das medidas da segunda avaliação experimental. 74
Tabela 15 – Matriz de Confusão da terceira avaliação experimental 75
Tabela 16 – Resultado do cálculo das medidas da terceira avaliação experimental. . 75
Tabela 17 – Matriz de Confusão com os resultados do quarto experimento. 75
Tabela 18 – Resultado do cálculo das medidas da quarta avaliação experimental. . . 76
Tabela 19 – Trabalhos selecionados no Mapeamento Sistemático de Estudos. 97

Lista de abreviaturas e siglas

A *Acurácia*

AM *Aprendizagem de Máquina*

API *Application Programming Interface*

APP *Applications*

CBIR *Content-Based Image Retrieval*

DOM *Document Object Model*

E *Erro*

FM *F-Measure*

FN *Falsos Negativos*

FP *Falsos Posiitivos*

GUI *Graphical User Interface*

IA *Inteligência Artificial*

IFN *Info-Fuzzy Networks*

IJCNN *International Joint Conference on Neural Networks*

IREP *Incremental Reduced Error Pruning*

JSF *JavaServer Faces*

LOST *Laboratory of Software Technology*

MCO *Mars Climate Orbiter*

MSE *Mapeamento Sistemático de Estudos*

P *Precisão*

QP *Questão de Pesquisa*

R *Recall*

ES *Especificidade*

RNA *Redes Neurais Artificiais*

ROC *Receiver Operating Characteristic*

SMS *Systematic Mapping Studies*

SQ *Subquestão de Pesquisa*

SUT *Software Under Testing*

SVM *Support Vector Machine*

URL *Uniform Resource Locator*

VN *Verdadeiros Negativos*

VP *Verdadeiros Positivos*

WEKA *Waikato Environment for Knowledge Analysis*

Sumário

I INTRODUÇÃO 1
Contexto e Motivação . . . **3**
Definição do Problema . . . **4**
Objetivos . . . **6**
Justificativa . . . **7**
Estrutura do Trabalho . . . **8**

II FUNDAMENTAÇÃO TEÓRICA 9

1 TESTES DE SOFTWARE . . . **11**
1.1 Níveis de Teste . . . **11**
1.2 Objetivos de Teste . . . **12**
1.3 Oráculos de Testes . . . **13**

2 APRENDIZAGEM DE MÁQUINA . . . **15**
2.1 Algoritmo IREP . . . **16**
2.1.1 Funcionamento do IREP . . . 16
2.2 Algoritmo AdaBoost . . . **16**
2.2.1 Funcionamento do AdaBoost . . . 17
2.2.2 Variações . . . 17
2.3 Medidas de Avaliação dos classificadores . . . **18**
2.3.1 Matriz de Confusão . . . 18
2.3.2 Acurácia – A . . . 18
2.3.3 Erro – E . . . 19
2.3.4 Precisão – P . . . 19
2.3.5 *Recall* – R . . . 19
2.3.6 Especificidade – ES . . . 19
2.3.7 *F-Measure* – FM . . . 20
2.3.8 Curva ROC . . . 20

3 TRABALHOS RELACIONADOS . . . **21**
3.1 Mapeamento Sistemático . . . **21**
3.1.1 Metodologia . . . 21
3.1.2 Questões de Pesquisa . . . 21
3.1.3 Processo de Busca . . . 22
3.1.4 Processo de Seleção de Estudos . . . 24

3.1.5 Processo de Extração . . . 26
3.1.6 Processo de Análise . . . 30
3.1.7 Resultados . . . 30
3.1.7.1 Resultados Gerais . . . 30
3.1.7.2 Resultados do Esquema de Classificação . . . 30
3.1.7.3 SQ1 - Técnica utilizada para automação dos oráculos de testes . . . 32
3.1.7.4 SQ2 - Tipo de oráculo . . . 34
3.1.7.5 SQ3 - Tipo de Teste de Software . . . 35
3.1.7.6 SQ4 - Restrições de Linguagem de Implementação . . . 36
3.1.7.7 SQ5 - Tipos de Softwares . . . 37
3.1.7.8 SQ6 - Método de Avaliação Empírica . . . 39
3.1.7.9 SQ7 - Tipo de software avaliado . . . 40
3.1.8 Discussão . . . 40
3.1.9 Ameaças à Validade . . . 43
3.2 Oráculos de Testes não incluídos no MSE . . . 44
3.2.1 Pseudo-Oráculos . . . 44
3.2.2 Testes Metamórficos . . . 45
3.3 Principais Trabalhos Relacionados . . . 46
3.4 Considerações Finais . . . 47

III PROPOSTA 49

4 ABORDAGEM PROPOSTA . . . 51
4.1 O Método . . . 51
4.1.1 Captura de Ações . . . 52
4.1.2 Preparação de Dados . . . 53
4.1.3 Aprendizagem de Máquina . . . 54
4.1.4 Avaliação da Aprendizagem . . . 54
4.2 Implementação do Método . . . 54
4.2.1 Implementação da Captura de Ações . . . 54
4.2.2 Implementação da Preparação de Dados . . . 56
4.2.3 Implementação da Aprendizagem de máquina . . . 57
4.2.3.1 Seleção de Classificador . . . 58
4.2.3.2 Preparação da Aprendizagem de Máquina . . . 59
4.2.4 Ferramenta . . . 60
4.2.4.1 Funcionalidades da Ferramenta . . . 61
4.3 Considerações Finais . . . 63

IV RESULTADOS E DISCUSSÕES 65

5 AVALIAÇÃO EXPERIMENTAL 67
5.1 Aplicações sob Teste 67
5.2 Questões de Pesquisa 68
5.3 Operação da Primeira Avaliação 68
5.4 Operação da Segunda Avaliação Experimental 71
5.5 Operação da Terceira Avaliação Experimental 71
5.6 Operação da Quarta Avaliação Experimental 72
5.7 Resultados das Avaliações Experimentais 73
5.8 Ameaças à Validade 76
5.8.1 Validade Externa 77
5.8.2 Validade Interna 77
5.8.2.1 Validade de Conclusão 77

6 CONCLUSÕES E TRABALHOS FUTUROS 79
6.1 Desafios e Limitações 81
6.2 Trabalhos Futuros 81

REFERÊNCIAS 83

APÊNDICES 95

APÊNDICE A – REFERÊNCIAS PARA TRABALHOS SELECIONADOS AO MAPEAMENTO SISTEMÁTICO . . 97

Parte I

Introdução

Contexto e Motivação

A computação tem sido utilizada para apoio à solução de problemas nas mais diversas áreas de conhecimento, como por exemplo, medicina, biologia, matemática, mecânica, administração, economia, etc. Inúmeras aplicações têm sido desenvolvidas para auxiliar o ser humano na realização de tarefas que antes eram realizadas de forma manual, tais como aplicações para identificação de anomalias em imagens médicas (medicina) (ANTONIE; ZAIANE; COMAN, 2001), aplicações para previsão de valores de ações no mercado financeiro (economia) (TRIPPI; TURBAN, 1992), aplicações para gestão patrimonial (administração) (MICHAUD; MICHAUD, 2008), dentre outras.

Entretanto, para garantir a confiabilidade das aplicações para solução de tais problemas é preciso que elas passem por uma etapa essencial no processo de desenvolvimento de software, o Teste de Software. O Teste de Software é uma das atividades que se concentra na gestão da qualidade de software (HORCH, 2003) e pode ser definido como um conjunto de tarefas, planejadas e executadas sistematicamente com o propósito de descobrir erros cometidos durante a implementação de um software (PRESSMAN, 1995).

Segundo Pressman (1995), a qualidade de software pode ser definida como uma gestão de qualidade efetiva, aplicada de modo a criar um produto útil, que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam, ou seja, é estabelecida uma infraestrutura que dá suporte à criação de um software que contemple as funções e recursos que o usuário deseja, garantindo confiabilidade, isenção de erros e que gere benefícios tanto para o usuário quanto para a empresa fabricante.

Estudos relatam que o Teste de Software é uma das mais atividades mais onerosas dentro do processo de desenvolvimento, podendo chegar até 50% do seu custo total (MYERS; SANDLER; BADGETT, 2011). Por conta disso, muitas vezes, empresas de desenvolvimento acabam negligenciando essa atividade de modo a priorizar a entrega dos produtos dentro do prazo estabelecido. No entanto, essa prática pode acabar causando prejuízos maiores tanto para a empresa desenvolvedora, quanto para o usuário final.

Como exemplo dos prejuízos causados por falhas consideradas simples e que não foram descobertas pela ausência de testes adequados, tome-se como exemplo o problema relacionado a Amazon.com, que durante um feriado do dia de ação de graças em 2001, sofreu uma série de interrupções em seus sistemas causando um prejuízo estimado em cerca de $25000 (vinte e cinco mil dólares) por minuto de inatividade (SOILA; NARASIMHAN, 2005).

Por conta do exposto, faz-se necessário o desenvolvimento de abordagens para dar suporte à atividade de Teste de Software, uma vez que essa atividade é propensa à erros, além de onerosa, se não tiver o apoio computacional. Dessa forma, o desenvolvimento dessas abordagens pode propiciar maior qualidade e produtividade da atividade de testes

e, consequentemente, redução do esforço necessário para o desenvolvimento de softwares com boa confiabilidade.

Felizmente, nos últimos anos, o número de pesquisas sobre automação em testes de software têm aumentado significativamente, como pode ser percebido por meio da Figura 1, obtida a partir de uma busca no *Scopus* utilizando o termo *"Automated Software Testing"*. Esse dado apresenta indícios de que a comunidade científica tem buscado alternativas de automação para as atividades de teste, em detrimento à sua forma tradicional, que ainda é intensamente baseada na figura humana.

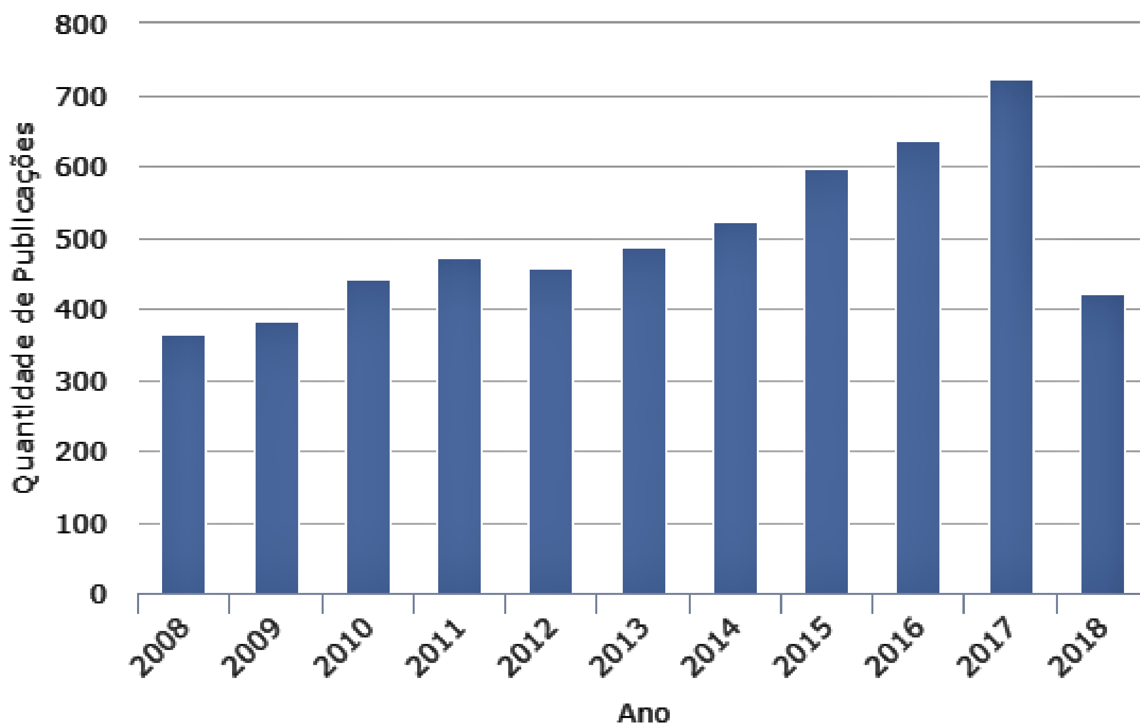


Figura 1 – Número de publicações sobre Teste de Software Automático obtidas no *Scopus*.

As técnicas baseadas em aprendizagem de máquina (AM) têm sido utilizadas para auxiliar a resolução de problemas complexos nas mais diversas áreas do conhecimento, como: modelagem de séries financeiras (PAVLIDIS et al., 2006), predição de eficácia de casos de teste (MAYRHAUSER; ANDERSON; MRAZ, 1995), estimar a importância de comentários sobre produtos e serviços (SANTOS et al., 2016), dentre outras. Estudos como esses apontam que a utilização de tais técnicas vêm obtendo bons resultados.

Definição do Problema

Considerando o contexto discutido, percebe-se que tem se tornado cada vez mais necessário o desenvolvimento de abordagens e ferramentas que tenham por objetivo a automação em atividades e mecanismos relacionados ao Teste de Software, reduzindo o

esforço e tempo necessário para a realização do processo e garantindo a qualidade do software desenvolvido (DELAMARO; NUNES; OLIVEIRA, 2013).

Um dos grandes desafios dessa área está relacionado a automação de um mecanismo conhecido como Oráculo de Teste. Dada uma determinada entrada para uma aplicação, a difícil tarefa de distinguir o comportamento correto para o comportamento potencialmente incorreto da aplicação é chamado de "Problema do Oráculo de Teste" (BARR et al., 2015). Geralmente, essa atividade é realizada por um desenvolvedor ou testador da aplicação, ou seja, figuras humanas, o que pode tornar essa tarefa bastante onerosa e propensa a erros, uma vez que, é necessário designar uma pessoa para inferir o resultado esperada de cada um dos testes de forma individualizada.

Existem uma série de dificuldades relacionadas à automação de um oráculo de teste. O primeiro desafio está relacionado em como prover o domínio de saídas dos softwares de forma automática, pois cada aplicação é construída com um objetivo diferente e identificar um conjunto de regras de negócio que possa representar as saídas dessas aplicações não é considerada uma tarefa trivial. A geração das saídas esperadas para cada aplicação é outra dificuldade dessa atividade, uma vez que muitas vezes elas são geradas manualmente baseadas na especificação de requisitos dos softwares. O terceiro desafio consiste na construção do mapeamento entre entradas e saídas da aplicação e, finalmente, o mecanismo que irá comparar saídas da aplicação e saídas esperadas para determinar se existe falha ou não em uma aplicação sob teste.

O Teste de Regressão é um tipo de teste realizado quando mudanças no software são implementadas. Esse tipo de teste tem por objetivo assegurar que as alterações realizadas não causaram defeitos no comportamento do software em relação à parte do software que não sofreu mudança. Dependendo do número de funcionalidades de uma aplicação, a realização de testes de regressão de forma que a cobertura dos testes atinga 100% dos cenários possíveis pode ser até inviável de ser realizada. Dessa forma, a utilização de um mecanismo (oráculo de teste automático) para monitoramento de um software em ambiente de produção pode auxiliar na identificação de falhas e redução de riscos associados à utilização do software pelos usuários.

A construção de um oráculo perfeito é inviável de ser construído (SANGWAN; BHATIA; SINGH, 2011). Caso existisse, poderia até substituir o software sob teste. Dessa forma, este trabalho visa uma nova abordagem com o objetivo de superar essas dificuldades e construir um mecanismo de oráculo de teste capaz de maximizar a identificação de falhas nos softwares em ambientes de produção.

Objetivos

Este trabalho tem como objetivo principal a criação de uma abordagem para apoio à realização de testes de software, por meio da automação do mecanismo de oráculo de teste. Para alcançar esse objetivo, a abordagem proposta contém quatro partes. A primeira, responsável pela obtenção de dados históricos das aplicações, é realizada utilizando uma técnica baseada na captura de *Logs* de execução das aplicações. A segunda etapa é destinada à preparação dos dados coletados para utilização na técnica de aprendizagem de máquina. Já a terceira etapa é responsável pela automação do oráculo de fato, a partir do uso de algoritmos de aprendizagem de máquina. Finalmente, na etapa quatro, são apresentados os relatórios de execução dos testes obtidos do oráculo automático implementado na etapa anterior. A abordagem proposta nesse trabalho é apresentada e discutida detalhadamente no Capítulo 4.

É importante destacar que a abordagem proposta neste trabalho pode ser utilizada para auxiliar à realização de Testes de Regressão em qualquer aplicação, desde que se tenha dados históricos do uso da aplicação em nível de cliente, que detalha as ações realizadas durante o uso, incluindo valores utilizados nos campos, elementos visíveis e comandos acionados. Esses dados serão utilizados como base para o uso da técnica de aprendizagem de máquina. Neste trabalho em específico, foca-se no uso de aplicações Web, uma vez que o uso de ferramentas já desenvolvidas no Laboratório LOST (SOUZA, 2016), torna simples capturar eventos do lado do cliente de forma não intrusiva.

Para alcançar o objetivo geral desse trabalho, foram definidos alguns objetivos específicos:

- Realizar um estudo das principais técnicas existentes na literatura utilizadas para automação do oráculos de testes em softwares, identificando as principais características, vantagens e limitações. Esse estudo foi a base para a definição da abordagem;

- Realizar um mapeamento sistemático sobre as técnicas e ferramentas para automação do oráculo de testes, para identificar os principais trabalhos da área e fornecer uma visão geral dessa linha de pesquisa. Esse estudo também foi importante para a tomada de decisões na definição da abordagem proposta, pois evidenciou lacunas na área de pesquisa deixada por outros trabalhos;

- Propor uma nova abordagem baseada em aprendizagem de máquina para auxiliar a automação do mecanismo de oráculo de testes;

- Avaliar a abordagem proposta, por meio da realização de avaliações empíricas. Os resultados preliminares obtidos na realização de provas de conceito apontam indícios da adequação ao problema.

Este trabalho está relacionado a um projeto maior com a intenção de aplicar teste de regressão de forma contínua em softwares em ambiente de produção. A ideia é construir uma abordagem para automação do mecanismo de oráculo de teste que possa ser utilizada de forma não intrusiva nas aplicações em ambiente real. O objetivo geral é identificar as falhas por meio da utilização de um mecanismo de monitoramento e alertar responsáveis sobre os problemas identificados para realização de devidas correções. Dessa forma, o objetivo geral deste trabalho é propor e avaliar a abordagem para automação do oráculo de teste.

Justificativa

Neste trabalho é proposta uma nova abordagem para auxiliar a realização de testes de softwares por meio da automação do oráculo de testes. Conforme discutido anteriormente, a etapa de Teste de Software é essencial para garantia de qualidade do software desenvolvido. Entretanto, existem alguns fatores que podem dificultar a realização do processo. Dentre eles, pode-se destacar:

- **Alto custo:** alguns estudos apontam que o teste de software pode consumir até 50% do esforço necessário em um projeto de desenvolvimento de software (MYERS; SANDLER; BADGETT, 2011). Desenvolvedores de software normalmente trabalham com prazos apertados para entrega de produtos. Assim, um mecanismo de apoie a realização de testes, reduzindo seu tempo de execução, é algo muito bem vindo para a área.

- **Realização manual propensa a erros:** a realização de testes de forma manual é propensa a erros, notadamente por parte dos desenvolvedores ou testadores, uma vez que normalmente essa atividade exige um grande esforço e também possui partes consideradas tediosas (DELAMARO; NUNES; OLIVEIRA, 2013). Assim, automatizar partes dessa atividade é fundamental para reduzir as chances de erro.

- **Avaliação de Resultados Tediosa:** após a implementação dos testes, uma etapa essencial é a avaliação dos resultados obtidos. Essa análise, geralmente realizada de forma manual, na qual os testadores ou desenvolvedores executam os testes e verificam o resultado de forma individualizada é uma tarefa tediosa e propensa à erros. É possível codificar a verificação de um resultado, mas toda vez que uma entrada for alterada, ou que o software seja alterado, todos os testes codificados precisam ser revistos, para confirmar se os resultados esperados continuam os mesmos. Isso encarece a manutenção de testes e torna a atividade tediosa.

- **Avaliação contínua em produção:** após a realização dos testes e implantação da aplicação em ambiente de produção. É de grande importância a avaliação do

comportamento da aplicação em um cenário real de forma contínua. Muitas vezes, podem ocorrer casos nos quais não foram previstos durante os testes quando não realizados de forma adequada, situações em que o software se comporte de maneira inesperada. A identificação desse comportamento em ambiente de produção irá reduzir riscos relacionados à prejuízos causados por falhas nos softwares.

A automação do mecanismo de Oráculo de Testes minimiza diretamente a influência desses fatores na realização do processo de Teste de Software, uma vez que, com a verificação do comportamento do Software Sob Teste (SUT, do inglês, *Software Under Testing*), de forma automática, existe uma redução no esforço dos desenvolvedores e, consequentemente, no custo da aplicação. Além disso, potencialmente pode-se reduzir falhas, pois a realização de forma manual pode ser tediosa e cansativa.

A automação de oráculos de testes em softwares tem sido uma área bastante explorada, conforme é apresentado e discutido no Capítulo 3. Entretanto, percebe-se que ainda existem diversas lacunas nessa linha de pesquisa, como por exemplo, a utilização de técnicas de aprendizagem de máquina para automação do oráculo de teste.

Estrutura do Trabalho

Este trabalho encontra-se dividido em seis capítulos e a introdução, que aborda o contexto do trabalho, sua motivação, objetivos geral e específicos, além do relato das principais contribuições e desta seção, com a estruturação do trabalho.

Nos capítulos 1 e 2 são descritos alguns dos conceitos fundamentais para uma boa compreensão do trabalho. O Capítulo 1 apresenta os conceitos principais relacionados ao Teste de Software e no Capítulo 2 são discutidos os conceitos relacionados ao Aprendizado de Máquina que fundamenta o desenvolvimento deste trabalho.

O Capítulo 3 descreve um Mapeamento Sistemático que tem por objetivo a identificação, análise e sumarização das pesquisas disponíveis na literatura referentes à automação de oráculos de testes.

Já no Capítulo 4 é apresentado um método e uma ferramenta proposto neste trabalho para automação do oráculo de teste. No Capítulo 5 são apresentados quatro experimentos realizados para avaliação da abordagem proposta. Finalmente, o Capítulo 6 apresenta as conclusões, limitações e perspectivas de trabalhos futuros da pesquisa.

Parte II

Fundamentação Teórica

1 Testes de Software

O Teste de Software é um subprocesso existente no processo de desenvolvimento de software. Muitas das vezes ele é mencionado como sendo um processo, subprocesso ou mesmo um processo guarda-chuva, indicando que esse conjunto de atividades é algo adicional a um grupo maior, que é o desenvolvimento de software. O Teste de Software também é considerado uma disciplina técnica no desenvolvimento de software, uma vez que aborda uma atividade bem focada, com conhecimento técnico específico associado.

O conceito de teste de software mais abrangente o descreve como sendo a verificação dinâmica do funcionamento de um programa utilizando um conjunto finito de casos de teste, adequadamente escolhido dentro de um domínio de execução infinito, contra seu comportamento esperado (BOURQUE; FAIRLEY et al., 2014). Segundo Pressman (1995), o teste de software consiste em um conjunto de tarefas, planejadas e executadas sistematicamente, com o propósito de descobrir erros em um software.

A realização de teste de software deve seguir um processo, que define os passos a serem executados, quando esses passos serão executados, por quem, além de detalhar tempo e recursos necessários. Os processos de teste de software devem incorporar planejamento dos testes, projeto de casos de teste, execução de testes, coleta e avaliação de resultados.

O teste de software é composto por um conjunto de elementos que normalmente causam confusão no seu entendimento e diferenciação. Um procedimento de teste consiste em uma documentação que especifica uma sequência de ações para execução de um teste. Os casos de testes são compostos por entradas, resultados esperados e um conjunto de condições de execução de um teste, além do detalhamento de qual procedimento de teste deve ser usado. A geração dos resultados esperados, contidos nos casos de teste, está intrinsecamente ligada a este trabalho. Um outro elemento bastante confundido no vocabulário de teste é Plano de Teste, que é um documento que descreve o escopo, abordagem, recursos e agenda para as atividades de teste, identificando os itens de teste, as implementações a serem testadas, as tarefas envolvidas, executores e riscos associados (IEEE, 1990).

1.1 Níveis de Teste

Levando-se em consideração o alvo de um teste, o teste de software pode ser classificado em diferentes níveis: testes de unidade, testes de integração e testes de sistema (BOURQUE; FAIRLEY et al., 2014).

O teste de unidade tem foco na verificação da menor unidade de um projeto de

software, por exemplo, componentes, módulos ou classes (PRESSMAN, 1995). Geralmente, esses testes são realizados pelos próprios desenvolvedores do software e construídos em paralelo com a implementação do software (PÁDUA, 2008). Seu objeto alvo são métodos, classes ou até pequenos trechos de código (NETO, 2007).

Os testes de integração têm por objetivo verificar o funcionamento do software após a integração das unidades, ou seja, o objetivo é descobrir erros associados às interfaces entre os módulos ou componentes de um sistema, quando esses elementos são integrados na construção da arquitetura do software que foi planejada (PRESSMAN, 1995).

Já o teste de sistema visa encontrar falhas no software por meio da utilização do software como um todo, como se fosse um usuário final. O software deve ser executado em condições similares às do usuário da aplicação, como, ambiente, dados de entrada e outras características (NETO, 2007). O teste de sistema também pode ser definido como uma série de diferentes testes cuja finalidade é executar totalmente o sistema, sob diferentes perspectivas, visando garantir seu adequado funcionamento sob todas essas visões diferentes (PRESSMAN, 1995).

1.2 Objetivos de Teste

Os testes também podem ser classificados em diferentes tipos levando-se em conta o objetivo do teste. Existem testes construídos com o objetivo de verificar se as especificações funcionais estão corretamente implementadas (teste funcional), podem ser construídos testes com o objetivo de verificar se o software está adequado ao uso (teste de usabilidade), para verificar se o software funciona sob condições anormais de demanda (teste de estresse), dentre outros.

Os testes funcionais são construídos com o propósito de verificar a consistência entre o produto implementado e os seus respectivos requisitos funcionais. Geralmente, realizado por uma equipe independente de teste, simulando a execução do software como um usuário final com o intuito de encontrar divergências entre o que foi implementado e o que foi requisitado. Esse objetivo de teste está diretamente ligado a esta proposta.

Os testes de regressão visam verificar se o software continua funcionando após uma alteração no código-fonte, ou seja, tem o propósito de verificar se o software não regrediu. Este objetivo de teste também está diretamente ligado a esta proposta.

Os objetivos citados não constituem todos os objetivos existentes, nesse trabalho foram apresentados apenas alguns conceitos básicos que fundamentaram a proposta dessa pesquisa.

1.3 Oráculos de Testes

Na literatura são encontradas diversas definições para oráculos de Testes. Segundo Barr et al. (2015) um oráculo de teste é um procedimento que realiza a distinção entre o comportamento adequado e o incorreto de um software sob teste. De acordo com Machado (2000) e Hamlet (1995), o oráculo de teste pode ser definido como um procedimento de decisão e estratégias para interpretação dos resultados de teste.

Após a execução de um determinado caso de teste em um SUT, é necessário coletar as saídas do SUT e identificar se as saídas estão corretas ou não para assim determinar a corretude do comportamento do software. A distinção entre o comportamento correto do incorreto pode ser determinada por meio da comparação entre as **saídas esperadas** e as **saídas atuais** do SUT.

Contudo, a criação de um oráculo de teste automático não é uma tarefa trivial. Segundo Oliveira, Kanewala e Nardi (2015), a primeira dificuldade consiste em identificar as saídas esperadas de um software, por exemplo: muitas vezes a identificação de saídas esperadas deve ser realizada com uma precisão decimal, outros podem realizar operações não-determinísticas e assim produzir diferentes saídas para uma mesma entrada. De acordo com Weyuker (1982), o problema do oráculo consiste em um conjunto de casos nos quais usar técnicas tradicionais é muito difícil ou impossível identificar a corretude de determinado caso de teste ou saída de um software.

Na literatura são encontrados diferentes tipos de oráculos de testes, levando em consideração a forma utilizada para se avaliar se o software se comportou de maneira adequada ou não. Existem oráculos baseado em modelo (LIU et al., 2016; CALVI; VIGANÒ, 2016; HILLAH et al., 2016), oráculos baseado em *logs* (DEYAB; ATAN, 2015; GAO; FANG; MEMON, 2015; ELYASOV et al., 2015), oráculos baseados em aprendizagem de máquina (JAMEEL; MENGXIANG; CHAO, 2016; SINGHAL; BANSAL; KUMAR, 2016; YOUSIF; SHAHAMIRI; MUSTAFA, 2015), pseudo-oráculos (DAVIS; WEYUKER, 1981), dentre outros. Cada um usa uma abordagem diferente, brevemente comentada a seguir.

Os oráculos de testes baseados em modelos, realizam a automação do oráculo de teste utilizando artefatos de documentação do software, que descrevem seu comportamento, visando com isso possibilitar a construção de um mecanismo automático para identificar comportamentos indevidos. Já oráculos baseados em *logs* são oráculos que utilizam a comparação de *logs* de execução entre diferentes versões do software para identificação de problemas. Os oráculos automáticos baseados em aprendizagem de máquina, usam dados históricos dos softwares para utilização em métodos computacionais capazes de fazer predições e auxiliar na tomada de decisões sobre o comportamento do SUT. O desenvolvimento de um Pseudo-Oráculo está relacionado a implementação de diferentes versões de um mesmo software que são utilizadas em paralelo para comparação de saídas

e identificação de problemas.

No Capítulo 3 é apresentado um Mapeamento Sistemático de Estudos com o objetivo de identificar na literatura os principais trabalhos que realizam a automação de oráculos de testes. São discutidas, dentre vários outros pontos, as principais técnicas utilizadas por pesquisadores da área para apoio à realização dessa atividade de forma automática.

2 Aprendizagem de Máquina

De acordo com Luger (2004), a aprendizagem é uma das tarefas, essencialmente humana, mais difícil de ser automatizada, pois ela engloba muitas outras habilidades inteligentes. A aprendizagem de máquina pode ser definida como uma sub-área da Inteligência Artificial (IA) com o objetivo de desenvolver técnicas computacionais inspiradas no aprendizado humano, assim como o desenvolvimento de sistemas capazes de adquirir conhecimento de forma automática.

Para Simon (1983), o aprendizado pode ser definido como qualquer mudança em um sistema que melhore seu desempenho ao realizar uma tarefa pela segunda vez, ou uma tarefa de mesmo domínio. De fato, a aprendizagem de máquina envolve aspectos na generalização a partir da experiência adquirida por meio da realização de tarefas anteriores, utilizando exemplos diferentes pertencentes à um mesmo domínio.

Existem diversas abordagens de aprendizagem de máquina que englobam diferentes algoritmos. São elas: **abordagens simbólicas**, **redes neurais ou conexionistas**, **aprendizado genético e evolucionário** (LUGER, 2004). As abordagens simbólicas são construídas sob a suposição de que a influência primária sobre o comportamento do programa é a base de conhecimento do domínio representada explicitamente, ou seja, a aprendizagem de máquina constrói ou modifica expressões em uma linguagem formal e retém o conhecimento para uso no futuro (LUGER, 2004).

As redes neurais ou conexionistas não adquirem conhecimento com base em sentenças simbólicas, inspiradas no sistema nervoso dos seres vivos. Elas possuem a capacidade de aquisição e manutenção do conhecimento baseado na organização e interação entre um conjunto de unidades de processamento chamadas de neurônios artificiais, que são interligados por um grande número de interconexões (sinapses artificiais) (SILVA; SPATTI; FLAUZINO, 2010).

Assim como as redes neurais, o aprendizado genético e evolucionário possui uma série de analogias biológicas que culminaram na concepção de uma série de algoritmos de aprendizagem de máquina. O poder de seleção natural de indivíduos em uma população tem sido demonstrado pela evolução das espécies em interação com o ambiente. Esse processo também tem sido refletido em pesquisas computacionais em algoritmos genéticos, programação genética e vida artificial (LUGER, 2004).

As técnicas de aprendizagem de máquina podem adquirir conhecimento por diferente formas de aprendizado: **supervisionado** e **não-supervisionado**. No aprendizado supervisionado o conjunto de exemplos apresentados é acompanhado por *"rótulos"*, que representam as classes a que eles pertencem, ou seja, existe uma classificação de cada

exemplo apresentado para treinamento. As novas ocorrências são classificadas com base no conhecimento adquirido pelo treinamento com exemplos apresentados anteriormente. Já no aprendizado não-supervisionado, não existe uma classe pré-definida para os exemplos apresentados. Nessa forma de aprendizagem, as técnicas de aprendizado de máquina analisam os exemplos apresentados e tentam identificar padrões e determinar se esses dados podem ser agrupados de alguma forma.

Apresentados os conceitos iniciais que serviram de base para o desenvolvimento deste trabalho, as Seções 2.1 e 2.2 apresentam os algoritmos de aprendizagem de máquina utilizados na construção da proposta da pesquisa.

2.1 Algoritmo IREP

O algoritmo IREP (do Inglês, *Incremental Reduced Error Pruning*) é um algoritmo baseado em regras, portanto, inserido na categoria de abordagens simbólicas de aprendizagem máquina e foi proposto por Fürnkranz e Widmer (1994). Após sua publicação, novas pesquisas foram realizadas com o objetivo de melhorar seu desempenho e solucionar problemas de generalização (COHEN, 1995).

2.1.1 Funcionamento do IREP

O algoritmo IREP cria um conjunto de regras de forma gulosa, uma por vez. Após a criação de uma regra, todos os exemplos cobertos pela regra são excluídos. O processo é repetido até que não haja mais exemplos positivos, ou até que a regra encontrada pelo IREP tenha uma taxa de erro excessivamente alta.

Para construção de uma regra, o IREP usa a seguinte estratégia. Inicialmente os exemplos são divididos em dois conjuntos, um conjunto crescente e um conjunto de podas. Após a criação dos conjuntos é criado um conjunto de condições de forma que maximize o ganho de informação até que a regra não cubra exemplos do conjunto crescente. Após a construção da regra, ela é então podada. O processo é repetido até que o conjunto final de regras seja formado (COHEN, 1995).

2.2 Algoritmo AdaBoost

O algoritmo *AdaBoost* é um dos algoritmos mais conhecidos da família de algoritmos de *Boosting*. Publicado pela primeira vez em (FREUND; SCHAPIRE, 1995), o *AdaBoost* pode ser utilizado em conjunto com qualquer algoritmo de aprendizagem de máquina para reduzir a taxa de erro dos algoritmos. Segundo Freund, Schapire et al. (1996), a técnica de *Boosting* é um método para aumentar o desempenho dos algoritmos de aprendizagem de máquina em relação à taxa de erro. Portanto, os algoritmos pertencentes à essa classe não

podem ser utilizados de forma isolada, mas sim em conjunto com algoritmos base, como algoritmos baseados em Redes Neurais Artificiais, Árvores de Decisão e outros.

O algoritmo resolveu alguns problemas existentes e apresenta algumas propriedades que facilitaram a sua implementação na prática em relação à seus algoritmos predecessores(FREUND; SCHAPIRE et al., 1996). As principais vantagens que podem ser destacadas são: facilidade de implementação, baixo valor computacional e possibilidade de utilização junto a diferentes classificadores base (FREUND; SCHAPIRE et al., 1996; SCHAPIRE; SINGER, 1998; FREUND, 2001).

O *AdaBoost* tem sido utilizado na resolução de problemas em diversas áreas de conhecimento, por exemplo, aprimoramento da sensibilidade e precisão de sensores (Engenharia Mecânica) (CHAVES, 2011), predição de gênero musical e artista a partir de arquivos de audio (Música) (BERGSTRA et al., 2006), descoberta de padrões no diagnóstico de doenças coronarianas (Medicina) (ZHAO et al., 2011), previsão da velocidade de vento em curtos prazos (Engenharia Elétrica) (WU; ZHANG; WANG, 2012), dentre outras.

2.2.1 Funcionamento do AdaBoost

Na primeira fase de funcionamento do algoritmo é realizado o treinamento do *AdaBoost*, na qual é apresentado um conjunto de exemplos $S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, onde: xi representa os parâmetros de entrada do sistema e yi representam as saídas esperadas para cada exemplo apresentado. A partir disso, o algoritmo base utilizado junto ao *AdaBoost* é chamado em uma série repetitiva de rodadas. A cada rodada uma distribuição de pesos referentes a cada exemplo é fornecida pelo *AdaBoost* ao algoritmo base.

A cada iteração o algoritmo base gera uma hipótese h_t que tem baixa taxa de erro em relação à distribuição de pesos atuais. Usando a hipótese gerada, o *AdaBoost* gera uma nova distribuição de pesos e o processo se repete. Ao final de T iterações a hipótese final h_f é gerada. Essa hipótese final combina as saídas de T hipóteses fracas (FREUND; SCHAPIRE, 1995).

2.2.2 Variações

Após ter sido proposto por Freund e Schapire (1995), diversas variações e extensões do algoritmo foram propostas por diversos pesquisadores, desenvolvidas com o objetivo de se obter os melhores resultados na solução de diferentes problemas.

Em sua versão original o algoritmo pode ser utilizado para resolução de problemas do tipo binário, ou seja, apresentam apenas duas classes para classificação dos exemplos. Na literatura são encontradas diversas variações para resolução de problemas dessa ordem, como, *BrownBoost* (FREUND, 2001), *Modest AdaBoost* (VEZHNEVETS; VEZHNEVETS, 2005) e outras.

Além disso, foram desenvolvidas extensões do algoritmo para auxiliar a resolução de problemas do tipo multi-classes, ou seja, problemas nos quais existem mais de duas classes para classificação dos exemplos. Pode-se destacar *AdaBoost.M1* (FREUND; SCHAPIRE et al., 1996), *AdaBoost.M2* (FREUND; SCHAPIRE et al., 1996), *AdaBoost.MR* (SCHAPIRE; SINGER, 1998), *AdaBoost.MH* (SCHAPIRE; SINGER, 1998)

2.3 Medidas de Avaliação dos classificadores

Nesta seção são apresentados os conceitos relacionados às medidas comumente utilizadas por trabalhos encontrados na literatura para avaliação do desempenho de classificadores. Para facilitar o entendimento dos conceitos, são apresentadas medidas e exemplos para avaliação de classificadores em problemas binários, no entanto, todas as medidas também podem ser aplicadas em problemas multi-classes (SOKOLOVA; LAPALME, 2009).

2.3.1 Matriz de Confusão

A matriz de confusão nada mais é que uma tabela para organização dos resultados obtidos da classificação realizada pelos algoritmos. Seus dados são preenchidos com o objetivo de comparar a classificação do algoritmo com o resultado esperado. Na Tabela 1 é apresentado um exemplo de uma matriz de confusão para problemas binários.

Tabela 1 – Exemplo de uma matriz de confusão para problemas binários.

	x	y
x	Verdadeiros Positivos	Falsos Positivos
y	Falsos Negativos	Verdadeiros Negativos

Os valores de x e y representam as classes hipotéticas do problema. Os valores para **Verdadeiros Positivos (VP)** indicam a quantidade de elementos da classe x que realmente foram classificados como x. Os **Falsos Positivos (FP)** indicam a quantidade de elementos da classe y que foram classificados como x. Já os **Falsos Negativos (FN)** indicam a quantidade de elementos da x, mas que foram classificados como y. Finalmente, os valores para **Verdadeiros Negativos (VN)** representam os elementos da classe y que realmente foram classificados como y.

2.3.2 Acurácia – A

A Acurácia de um classificador representa a porcentagem de exemplos corretamente classificados pelo algoritmo utilizado. Por meio dessa medida é possível determinar a qualidade do classificador de forma geral, ou seja, calcula a porcentagem de acertos em

relação ao total de elementos. A Acurácia de um classificador em problemas binários pode ser calculada por meio da Equação 2.1.

$$A = \frac{VP + VN}{VP + FP + FN + VN} \tag{2.1}$$

2.3.3 Erro – E

O valor da taxa de erro na avaliação do desempenho, representa a porcentagem de elementos classificados incorretamente. Essa medida pode ser calculada pelo complemento da Acurácia, conforme apresentada na Equação 2.2

$$E = 1 - A \tag{2.2}$$

2.3.4 Precisão – P

A precisão é a medida que calcula a porcentagem de elementos verdadeiros positivos em relação ao total de elementos classificados como positivos. Essa medida mostra o desempenho do classificador em relação a cada uma das classes dos problema. A precisão pode ser calculada por meio da Equação 2.3

$$P = \frac{VP}{VP + FP} \tag{2.3}$$

2.3.5 *Recall* – R

O *recall* ou sensitividade é a medida que infere a porcentagem de verdadeiros positivos em relação a soma dos elementos da classe. Assim, é possível identificar a proporção de elementos de uma classe x em relação a todos que deveriam ter sido classificados como x. O *recall* pode ser calculado por meio da Equação 2.4.

$$R = \frac{VP}{VP + FN} \tag{2.4}$$

2.3.6 Especificidade – ES

A especificidade é a medida que calcula a porcentagens de amostras negativas identificadas corretamente em relação ao total de amostras negativas. Essa medida é calculada por meio da Equação 2.5.

$$ES = \frac{VN}{VN + FP} \tag{2.5}$$

2.3.7 *F-Measure* – FM

A medida *F-Mesasure* combina as medidas de precisão e *recall*, seu valor é aproximadamente a média dessas medidas quando seus valores estão próximos. O resultado do cálculo dessa medida é dado no intervalo entre 0 e 1, quanto mais próximo de 1 melhor o classificador. O cálculo da *F-Measure* pode ser realizado por meio da Equação 2.6.

$$FM = \frac{2 * P * R}{P + R} \tag{2.6}$$

2.3.8 Curva ROC

A curva ROC (do Inglês, *Receiver Operating Characteristic*) é a representação gráfica relacionando sensitividade (*recall*) e especificidade. Os pontos da curva representam o desempenho de um classificador em uma determinada distribuição. A Figura 2 apresenta um exemplo de uma curva ROC. O ponto (0,1) representa o classificador perfeito, todas as amostras positivas e negativas classificadas corretamente, o ponto (0,0) representa um classificador que classifica todas as amostras como negativas, já o ponto (1,1) classifica todas as amostras como positivas e o ponto (1,0) representa um classificador que classifica todas as amostras incorretamente.

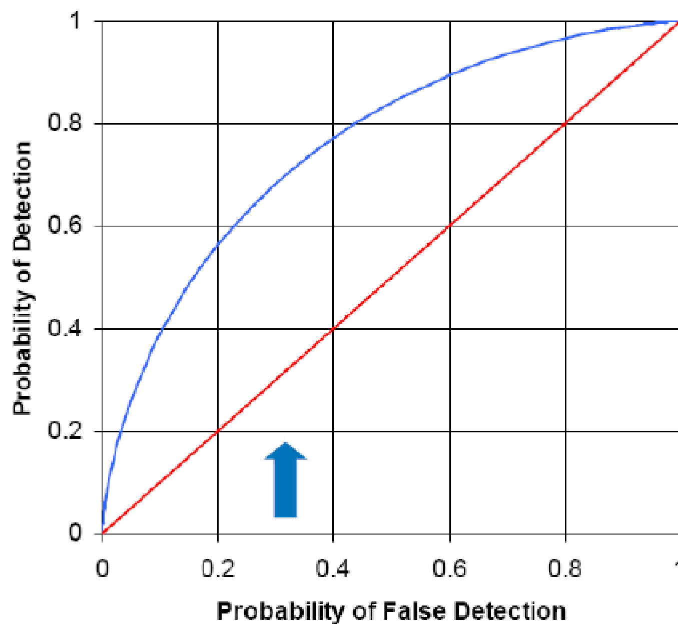


Figura 2 – Exemplo de uma Curva ROC.

3 Trabalhos Relacionados

Na literatura são encontrados alguns trabalhos que tem o objetivo de automatizar o mecanismo de Oráculo de Testes. Este capítulo discute os trabalhos identificados por meio de um Mapeamento Sistemático, além de alguns trabalhos identificados por meio de uma pesquisa não sistematizada sobre o tema. Por fim, são apresentadas considerações que ajudam a posicionar este trabalho dentro dessa linha da pesquisa.

3.1 Mapeamento Sistemático

3.1.1 Metodologia

Foi escolhido a metodologia de Mapeamento Sistemático de Estudos (MSE) para identificar e analisar o estado da arte em relação à automação em oráculos de testes em softwares. O MSE é um método científico capaz identificar, interpretar e sumarizar os trabalhos relevantes para determinada linha de pesquisa, área ou fenômeno de interesse de forma não tendenciosa e replicável (KEELE, 2007). Essa metodologia utiliza estudos primários relativos a uma Questão de Pesquisa (QP) com o objetivo específico de integrar/sintetizar as evidências relacionadas a essa questão.

Esse método teve origem na Medicina e foi desenvolvido para que os médicos pudessem se manter atualizados por meio de um acompanhamento da evolução das pesquisas científicas relacionadas ao tratamento de certas patologias (HIGGINS; GREEN et al., 2008). Entretanto, devido a sua eficácia, ele passou a ser aplicado em diversas linhas de pesquisa, inclusive ligadas a Ciência da Computação, como Engenharia de Software, Inteligência Artificial, dentre outras (PETERSEN et al., 2008).

A pesquisa foi realizada seguindo as diretrizes propostas em (KEELE, 2007) e implementado parcialmente o processo proposto por Petersen et al. (2008). O processo de mapeamento aplicado é sumarizado na Figura 3 e detalhado nas subseções 3.1.2-3.1.6

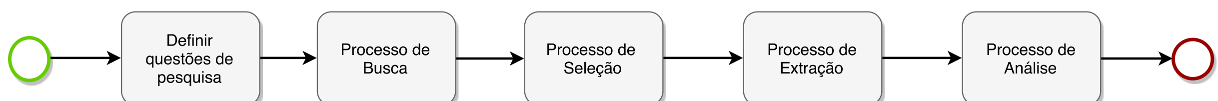


Figura 3 – Processo do MSE.

3.1.2 Questões de Pesquisa

Para alcançar os objetivos do MSE foi definida a seguinte questão de pesquisa:

Questão de Pesquisa (QP) - Quais os métodos ou ferramentas existentes automatizam o oráculo de testes? Esta questão motivou a elaboração das seguintes subquestões:

- **SQ1 -** Qual a técnica utilizada para automação do oráculo?
- **SQ2 -** Qual o tipo de oráculo implementado?
- **SQ3 -** Quais tipos de testes de software podem ser realizado com apoio do método/ferramenta?
- **SQ4 -** Existem restrições de linguagem de implementação?
- **SQ5 -** Quais tipos de softwares podem ser testados com apoio do método/ferramenta?
- **SQ6 -** Como o método/ferramenta foi avaliado?
- **SQ7 -** A avaliação foi realizada com softwares reais?

O objetivo da QP consiste na identificação dos estudos primários que descrevam um método ou ferramenta para apoio à automação do oráculo de teste. A SQ1 tem foco na identificação da técnica utilizada para auxiliar a automação do oráculo de teste. A SQ2 foi adaptada de (BARR et al., 2015) para identificação do tipo de oráculo proposto baseando-se nos artefatos utilizados para automação do oráculo de testes. Com a SQ3 é realizada a identificação dos tipos de testes de softwares nos quais o método/ferramenta pode ser aplicado. Já a SQ4 foi elaborada com o objetivo de identificar se os oráculos de testes possuem restrições de uso relacionadas à linguagem de implementação do SUT. A SQ5 visa descobrir os tipos de softwares que podem ser testados com auxilio do método/ferramenta. Já a SQ6 tem como foco a identificação do método de avaliação empírica utilizado para avaliar cada trabalho. Finalmente, a SQ7 visa descobrir como a avaliação do trabalho foi realizada, se foi avaliado softwares reais, sejam eles industriais ou acadêmicos, ou foi avaliado com *toy programs*, programas fictícios, que não estão disponíveis em ambiente industrial.

3.1.3 Processo de Busca

O processo de busca de estudos primários é apresentado na Figura 4 e tem seis atividades.

Inicialmente foram definidos os termos que seriam incluídos na *string* de pesquisa. Os termos incluídos foram “method”, “automated test oracle”. Após a identificação dos termos chaves para a busca de estudos também foram incluídos algumas palavras que poderiam substituir os termos. Os seguintes sinônimos foram incluídos na string de pesquisa:

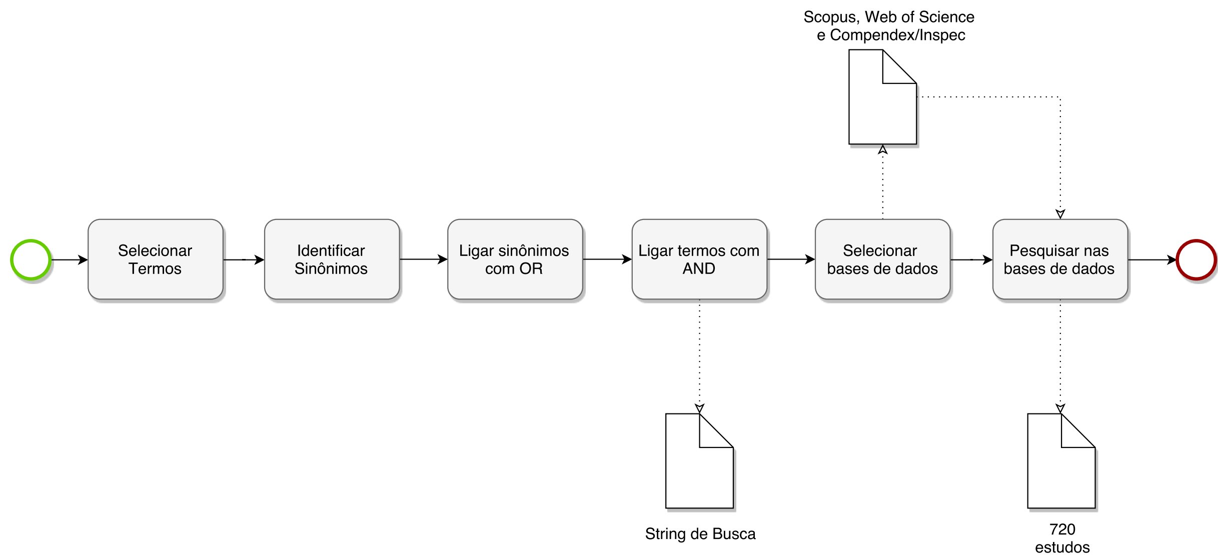


Figura 4 – Processo de Busca do MSE.

- **Method -** methodology, approach, technique, tool, framework, application, API, library;

- **Automated test oracle -** automatic test oracle;

Os termos sinônimos foram ligados ao termo chave usando o conectivo "OR" e posteriormente, as *strings* resultantes foram ligadas utilizando o conectivo "AND". A *string* de pesquisa final é apresentada a seguir:

(*tool* **OR** *method** **OR** *application* **OR** *approach** **OR** *techn** **OR** *framework* **OR** *API* **OR** *librar**) **AND** (*autom* test* oracle**)

Após a formulação da string de pesquisa, foram selecionadas as bases de dados para busca dos estudos relevantes. Scopus[1], Web of Science[2] e Engineering Village[3] foram as bases escolhidas porque elas realizam indexação das bases de dados mais relevantes no que diz respeito à Engenharia de Software, como IEEExprole, Springer, ACM e Elservier. A string foi aplicada para buscas por metadados (título, resumo e palavras-chave) nas bases selecionadas limitando o escopo da busca por trabalhos publicados período de 2000 à 2016.

Um problema comum durante a realização de mapeamentos sistemáticos é a presença de trabalhos duplicados. Foram removidos artigos com título igual ou que possuíam a distância de Levenshtein (LEVENSHTEIN, 1966) abaixo de um limiar de quatro caracteres de diferença em relação aos demais trabalhos. O processo de remoção

[1] Scopus WebSite: <https://www.scopus.com/>
[2] Web of Science WebSite: <https://www.webofknowledge.com/>
[3] Engineering Village WebSite: <https://www.engineeringvillage.com/>

de trabalhos duplicados foi realizado com suporte da ferramenta TheEnd[4]. A Tabela 2 apresenta o resultado da pesquisa realizada em cada base de dados.

Tabela 2 – Resultado da busca nas bases de dados.

Base de Dados	**Resultados**
Engineering Village	381
Scopus	480
Web Of Science	306
Total	1167
Total sem duplicações	720

3.1.4 Processo de Seleção de Estudos

O processo de seleção de trabalhos é apresentado na Figura 5 e foi dividido em cinco atividades.

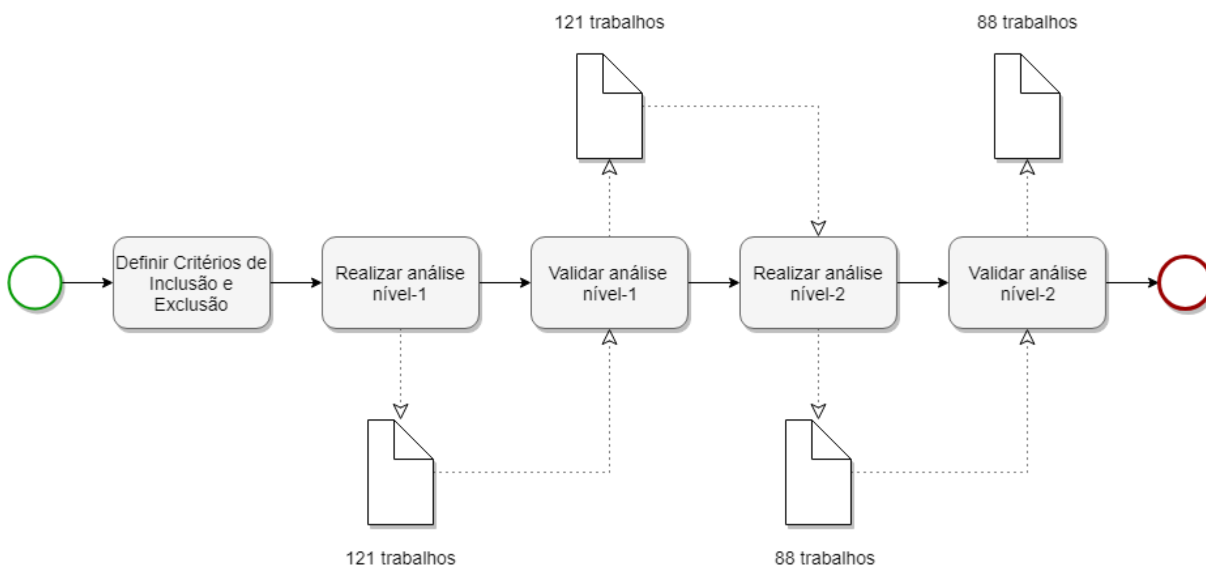


Figura 5 – Processo de Seleção de Estudos.

Inicialmente, foram definidos os seguintes critérios de inclusão e exclusão em conformidade com os objetivos e questões de pesquisa do MSE:

- **Critérios de Inclusão** Decidiu-se que seriam incluídos no MSE estudos que:

  A. Fossem escritos em Inglês **AND**;

  B. Tivessem sido publicados em *journal*, conferências, workshops no período de 2000 à 2016 **AND**;

  C. Apresentassem um método/ferramenta que tenha como objetivo a automação do oráculo de testes em softwares.

[4] TheEnd WebSite: <http://easii.ufpi.br/theend>

- **Critérios de Exclusão** Decidiu-se que seriam excluídos do MSE os estudos que:

  A. Não fossem escritos em Inglês **OR**;

  B. Não tivessem sido publicados em *journal*, conferências, workshops no período de 2000 à 2016 **OR**;

  C. Não apresentassem um método/ferramenta que tenha objetivo a automação do oráculo de testes em softwares **OR**;

  D. Estivessem disponíveis somente em forma de *abstract*, apresentação ou resumo expandido.

O processo de seleção de estudos primários foi conduzido em duas etapas de análise. Na primeira etapa (Análise Nível-1), foram considerados apenas título e resumo dos trabalhos. Na segunda etapa (Análise Nível-2), o texto completo dos trabalhos foram lidos. É importante ressaltar que em ambas as etapas foi aplicada uma abordagem inclusiva, isto é, em caso de trabalhos duvidosos decidiu-se pela inclusão do trabalho.

Na Análise Nível-1, os 720 trabalhos encontrados após a aplicação da strings de busca, foram igualmente divididos entre dois pesquisadores (A e B), como resultado dessa etapa, 121 trabalhos foram julgados como relevantes para o MSE. Para aumentar a confiabilidade da seleção dos trabalhos, 5% dos trabalhos analisados pelo Pesquisador A foram selecionados aleatoriamente para análise do Pesquisador B e vice-versa, após a leitura foi calculado o índice Kappa(COHEN, 1960) tendo como resultado 0.98, classificando a concordância entre os pesquisadores como "quase perfeita". Além disso, os trabalhos em que houve discordância foram novamente avaliados e discutidos.

Na etapa de análise nível-2, cada um dos 121 trabalhos restantes foi lido por dois pesquisadores (A e B), tendo sido aceitos 88 trabalhos como estudos primários relevantes ao MSE. Para tentar garantir a confiabilidade da seleção realizada, o processo foi realizado por dois pesquisadores e o trabalhos nos quais houve discordâncias foram novamente avaliados e discutidos com auxilio de um terceiro participante (Pesquisador C).

Os critérios utilizados para excluir os trabalhos, utilizados durante o processo de seleção (análise nível 1 e 2), são apresentados na Tabela 3

Tabela 3 – Critérios para exclusão de trabalhos.

Critério	**Resultados**
Trabalhos duplicados	447
Não apresentar método/ferramenta para automação do oráculo de testes	253
Texto completo não acessível	19
Trabalhos não escritos em inglês	1

3.1.5 Processo de Extração

O processo de extração de dados é apresentado na Figura 6 e consiste em quatro etapas principais: definir um esquema de classificação, definir um formulário de extração, extrair os dados e validar os dados extraídos.

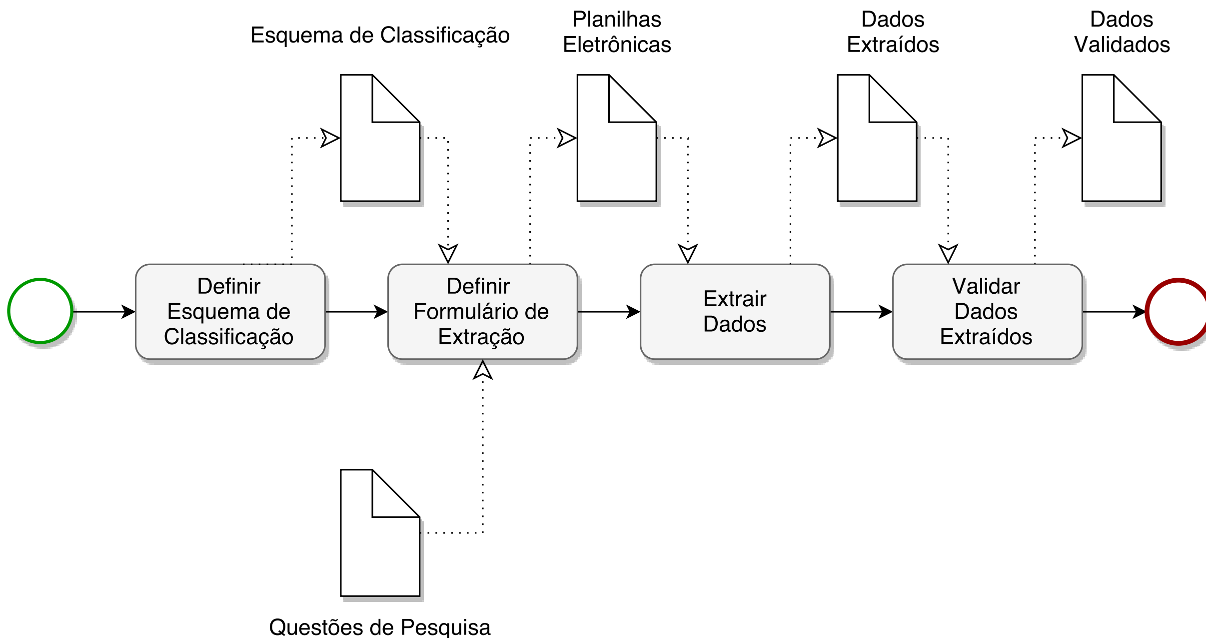


Figura 6 – Processo de extração de dados.

Os dados extraídos dos trabalhos selecionados foram organizados em planilhas contendo alguns campos a serem preenchidos. Os campos Identificador do Estudo, Título, Ano, Tipo de Evento, Local de Publicação, Autores foram extraídos automaticamente dos arquivos *bibtex*. Além dessas informações foram extraídas algumas informações por meio da leitura dos trabalhos para criação do esquema de classificação, sendo elas:

- **Contribuição -** Esta questão foi adaptada de (SHAW, 2003) e refere-se ao tipo de contribuição proposta pelo trabalho, tendo sido classificada nas seguintes categorias:

  A. Modelo - Representação de uma realidade observada por conceitos ou conceitos relacionados depois de um processo de conceitualização;

  B. Framework/método - Os modelos relacionaram-se à construção de software ou processos de gerenciamento de desenvolvimento;

  C. Ferramenta - A tecnologia, o programa ou a aplicação costumaram criar, depurar, manter ou apoiar processos de desenvolvimento;

  D. Teoria - Construção de uma relação causa-efeito relacionadas aos resultados de uma pesquisa;

E. Guidelines - Consiste em uma lista de orientações, obtidas a partir de uma síntese dos resultados de uma pesquisa;
F. Lição - Conjunto de resultados, diretamente analisados dos resultados de pesquisa obtidos;
G. Conselho/Implicação - Recomendações discursivas e genéricas, consideradas a partir de opiniões pessoais.

- **Tipo de Pesquisa -** Esta questão é usada para distinguir os diferentes tipos de estudos primários (adaptada de (WIERINGA et al., 2006)) e foi classificada nas seguintes categorias:

A. Validação de Pesquisa - técnicas investigadas são novas e ainda não foram implementadas em larga escala industrial. A avaliação realizada é formal e envolve interação humana, mas os indivíduos e o ambiente utilizados são pouco representativos do contexto real. As técnicas utilizadas são, por exemplo, experimentos realizados com estudantes;
B. Pesquisa de Avaliação - técnicas foram implementadas na prática e avaliadas em larga escala industrial, experimentadas com profissionais ou aplicadas em outro contexto real. Mostra-se como a técnica é implementada na prática (a implementação da solução) e quais são as consequências da implementação em termos de vantagens e desvantagens de avaliação (implementação);
C. Proposta de Solução - a solução para o problema é proposto, sendo nova ou um aumento significativo de uma solução existente. Os potenciais benefícios e aplicabilidade da solução são mostrados apenas por um pequeno exemplo, simulação, prova de conceito ou em uma boa argumentação;
D. Trabalhos Filosóficos - esboçam uma nova maneira de olhar as coisas existentes, estruturando o campo na forma de uma taxonomia ou estrutura conceitual;
E. Trabalhos de Opinião - expressam a opinião pessoal de alguém se determinada técnica é boa ou ruim, ou como as coisas deveriam sido feitas, não dependem de metodologias de trabalho e de trabalhos relacionados;
F. Trabalhos de Experiência - explicam sobre o que e como algo foi feito na prática, retratam a experiência pessoal dos autores.

- **Técnicas de Automação para Oráculos de Teste -** Refere-se ao método utilizado para automatizar o oráculo de teste, este é um campo textual, alguns termos identificados em diversos trabalhos (DELAMARO; NUNES; OLIVEIRA, 2013) que podem ser utilizados para a classificação são:

A. Model-based – quando a abordagem utiliza a especificação do sistema, documento modelo ou diagramas para automatizar o oráculo de testes;

B. N-version - automatiza o oráculo por meio da utilização de versões anteriores do SUT;

C. Machine learning – utiliza técnicas de aprendizagem de máquina para automação do oráculo;

D. GrO-method – *Graphical Oracle method*, nomenclatura proposta em (DELAMARO; NUNES; OLIVEIRA, 2013) que utiliza técnica para automação do oráculo baseada em conceitos de CBIR (*Content-Based Image Retrieval*), ou seja, utiliza a similaridade entre imagens para comparação de saídas de um SUT;

E. Outras – outra técnica para automação (técnica informada no campo textual).

- **Tipo de Oráculo -** Utilizada para distinguir os diferentes tipos de oráculos de testes baseado nos artefatos utilizados para automação (adaptada de (BARR et al., 2015)), são classificados em:

  A. Specified – Quando o oráculo necessita da especificação formal do SUT, também são incluídos nessa categoria especificações parciais do sistema, como assertions e models. Muitos formalismos para testes baseados em especificação existem na literatura, dessa forma, são incluídos nesta categoria oráculos que utilizam as seguintes técnicas: *model-based specification languages*, *state transition systems*, *assertions and contracts* ou *algebraic specifications*;

  B. Derived – Distingue o comportamento correto de um sistema utilizando vários artefatos, por exemplo, documentação, execuções do sistema, propriedades do SUT, versões anteriores;

  C. Implicit - Um oráculo implícito depende de um conhecimento implícito do comportamento do SUT, um oráculo implícito pode ser construído a partir de um procedimento que detecte anomalias no comportamento do SUT. Oráculos implícitos não são universais, um comportamento anormal para um sistema em um contexto, pode ser normal para outro sistema em contextos diferentes;

- **Método de avaliação empírica -** Esta questão é utilizada para distinguir o método de avaliação utilizado no trabalho proposto, segundo WOHLIN et al. (2012) na Engenharia de Software esses métodos são classificados em:

  A. *Survey* - quando a avaliação da pesquisa é realizada por meio de questionários, que são respondidos por um grupo de participantes selecionados;

  B. Estudo de Caso - quando a investigação sobre o tema é realizada no contexto real em um determinado espaço de tempo;

  C. Estudo experimental - a pesquisa é realizada em ambiente controlado envolvendo mais de um tratamento para comparação dos resultados;

D. Prova de Conceito - consiste em experimentos iniciais realizados com o objetivo de demonstrar se uma determinada ideia é tecnicamente possível.

- **Tipo de Teste -** Questão formulada com o intuito de identificar em quais tipos de testes de softwares o método/ferramenta pode ser utilizado, pois muitos trabalhos propõem oráculos de testes aplicados em tipos específicos de teste de software, por exemplo, testes funcionais, testes de regressão, testes de segurança, testes de desempenho, dentre outros.

- **Possui restrições de linguagem -** (Sim/Não) Esta questão é utilizada para identificar se o método/ferramenta proposto em cada trabalho possui alguma restrição de utilização em softwares implementados em diversas linguagens de programação.

- **Tipo de software avaliado -** Esta questão é utilizada para distinguir trabalhos aplicados na indústria ou pesquisas teóricas. Para esta classificação são utilizadas as informações de avaliação do trabalho, são consideradas estudos de casos, provas de conceito, exemplos de uso, quase-experimentos, experimentos controlados e análises empíricas como experimentos. Esta vertente pode ser classificada em:

  A. Softwares Reais – quando o método proposto foi avaliado em softwares comercias ou que tenham publicações referenciando o software;

  B. Toy Programs – quando o método proposto foi avaliado somente em softwares em desenvolvimento, ou produzidos pelos próprios autores, sistemas acadêmicos, etc.

- **Tipo de Software -** Essa questão é utilizada para identificar os tipos de softwares nos quais cada oráculo de testes pode ser aplicado, por exemplo, jogos, APP's para smartphones e TV's, *Web-based softwares*, *GUI applications*, dentre outros.

O processo de extração de dados foi realizado por dois pesquisadores (A e B) de forma independente. Dessa forma, cada pesquisador leu todos os 88 trabalhos selecionados armazenando os dados em planilhas eletrônicas contendo as questões definidas nos esquema de classificação.

A imprecisão na extração de dados e os erros de classificação referem-se a possibilidade de extração de informações de um estudo de diferentes maneiras por pesquisadores diferentes. Para aumentar a confiabilidade do processo de extração de dados, foi realizada uma etapa de validação dos dados extraídos, na qual, os resultados obtidos por cada pesquisador foram comparados. Como resultado da validação, foram identificadas divergências parciais nos dados extraídos em 19 trabalhos, mas foram facilmente resolvidas após os pesquisadores entrarem em consenso.

3.1.6 Processo de Análise

No processo de análise foram identificados os principais conceitos de cada estudo primário, que foram organizados em forma de tabelas para permitir comparações entre os estudos e classificar categorias. A análise quantitativa foi baseada na contagem dos estudos primários que foram classificados em cada alternativa das sub-questões de pesquisa e apresentada na forma de gráfico. Por fim são discutidos os resultados referente a cada uma das questões de pesquisa.

3.1.7 Resultados

Nesta seção são apresentados os resultados do MSE realizado em relação aos dados extraídos dos 88 trabalhos selecionados que satisfazem às questões de pesquisa. Além disso, são discutidos os principais pontos da área a fim de identificar lacunas deixadas pelos trabalhos disponíveis na literatura em relação à cada uma das questões de pesquisa definidas nesse MSE.

É importante ressaltar que nesse MSE é realizada uma análise quantitativa da área. Assim, os resultados são apresentados e discutidos com relação ao número de trabalhos classificados para cada uma das questões de pesquisa. A classificação individual de cada trabalho selecionado está disponível em uma Planilha Eletrônica[5] e a Tabela 19 apresenta os ID's e referencias para os trabalhos selecionados nesse MSE.

3.1.7.1 Resultados Gerais

A Figura 7 apresenta o número de pesquisas selecionados nesse MSE com objetivo de automação de oráculos de testes em softwares. A figura mostra que o número de pesquisas tem crescido bastante nos últimos anos (principalmente no ano de 2015), que sugere um alto nível de interesse de pesquisadores nesta área de pesquisa.

A Tabela 4 mostra que 56% dos estudos tem sido publicados em conferências relevantes da área, 10% dos trabalhos selecionados foram publicados em simpósios, 29% dos estudos foram publicados em *journals* e 4% dos trabalhos foram publicados em workshops, evidenciando os principais tipos de locais de publicação de pesquisas científicas relacionadas à automação de oráculos de testes.

3.1.7.2 Resultados do Esquema de Classificação

Nesta seção serão apresentados os resultados correspondentes às facetas que compõem o esquema de classificação descrito na Seção 3.1.5.

A Figura 8 apresenta os resultados da classificação dos trabalhos quanto ao **Tipo de Pesquisa** (Eixo X) (WIERINGA et al., 2006) e **Contribuição** (Eixo Y) (SHAW,

[5] Resultados do MSL: <https://goo.gl/whcbdL>

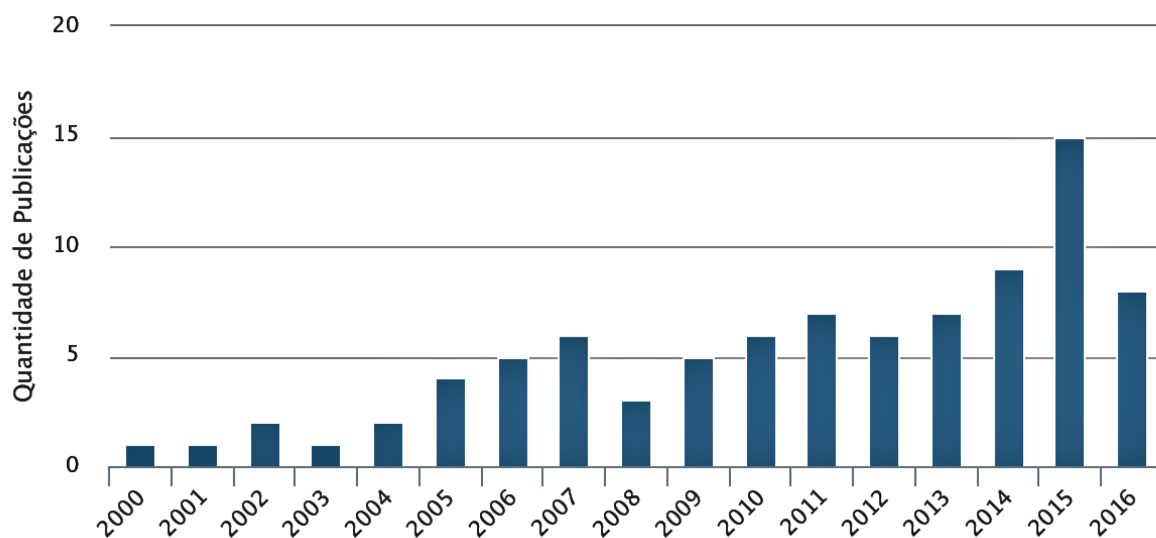


Figura 7 – Distribuição das publicações identificadas entre os anos de 2000 e 2016.

Tabela 4 – Locais de publicação com trabalhos aceitos no MSE.

Tipo de Evento	**Número de trabalhos**	**Porcentagem**
Conferência	49	56%
Simpósio	9	10%
Journal	26	29%
Workshop	4	5%

2003), analisando a figura pode-se perceber que 69% dos estudos primários (61 trabalhos) foram classificados como Proposta de Solução e *Framework/Method*, pois apresentam uma nova solução para o problema por meio da criação de um método ou uma abordagem para automação do oráculo de testes.

É importante destacar que não foram identificados trabalhos dos tipos Filosóficos, de Experiência ou de Opinião, além de trabalhos que visem propor Teorias, Lições, *Guidelines* e Conselhos. Apenas um trabalho encontrado foi classificado como Modelo em relação à sua Contribuição.

Ainda analisando a Figura 8, pode-se perceber que outros 23% dos estudos (21 trabalhos) visam propor uma Ferramenta para apoio à automação de oráculos de testes, além de dois trabalhos que realizam uma Pesquisa de Avaliação e seis trabalhos que realizam uma Validação de Pesquisa em ferramentas propostas em trabalhos anteriores dos autores.

Dessa forma, é possível perceber que grande parte das pesquisas está voltada ao desenvolvimento de novos métodos para automação de oráculos de testes, entretanto, tais métodos ainda foram pouco avaliados na indústria, apresentando indícios que existem lacunas na área de pesquisa.

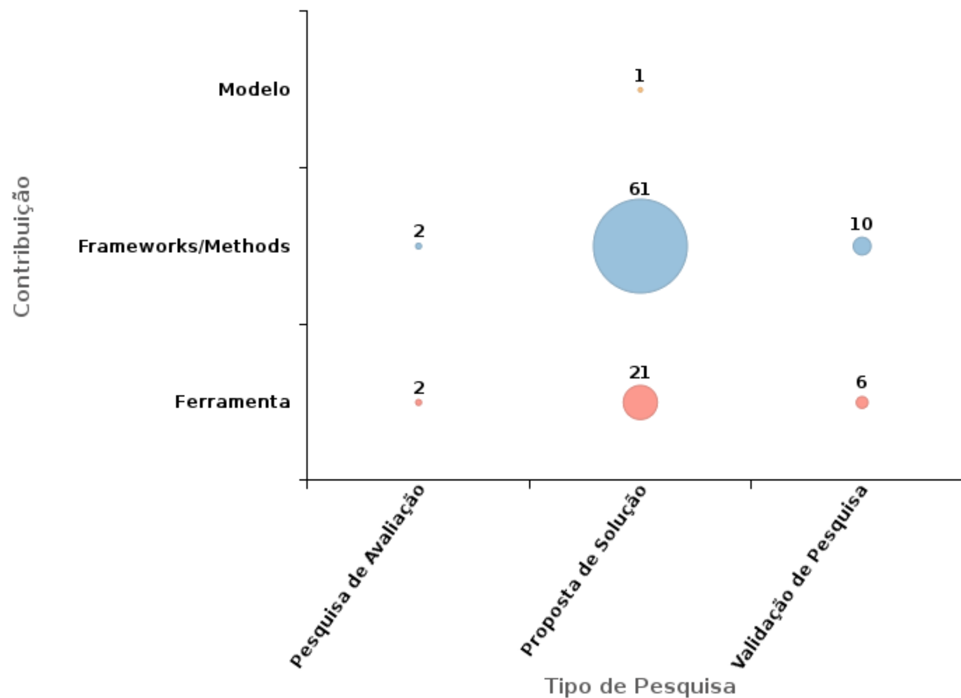


Figura 8 – Mapa Sistemático relacionando Contribuição e Tipo de Pesquisa.

3.1.7.3 SQ1 - Técnica utilizada para automação dos oráculos de testes

A Subquestão de pesquisa **SQ1** foi elaborada com o intuito de identificar na literatura as principais técnicas utilizadas para automação do mecanismo de oráculos de testes. Essa questão irá facilitar aos pesquisadores a identificação de métodos já existentes e assim auxiliar na criação de novas técnicas evitando que sejam feitas reinvenções dos métodos anteriores. Foram identificadas as seguintes técnicas de automação de oráculos de testes:

- **Model-based -** Essa técnica utiliza documentações do software, especificações de requisitos, diagramas de sequência, diagramas UML, dentre outros artefatos para automação dos oráculos de testes, ou seja, os resultados de execuções de testes em uma aplicação são comparados com os resultados obtidos a partir de modelos com o objetivo de identificar falhas no software.

- **Machine-learning -** Utiliza técnicas baseadas em aprendizagem de máquina para automação do mecanismo de oráculo de testes, por exemplo, Redes Neurais Artificiais, SVM (*Support Vector Machine*), IFN (*Info-Fuzzy Networks*), dentre outras.

- **Log-based -** Nessa técnica os pesquisadores realizam a automação dos oráculos de testes por meio da captura de logs de execução de uma aplicação, esses logs são então comparados em diferentes versões do software para identificação de falhas

na aplicação. Essa técnica é utilizada principalmente em testes de regressão, pois é necessário que seja realizada uma captura prévia dos logs da aplicação.

- **GrO-Method -** Esse método propõe a automação do oráculo baseado em propriedades de imagens extraídas durante a utilização de determinada aplicação. O método utiliza uma técnica conhecida como CBIR (do Inglês, *Content-Based Image Retrieval*) utilizada para busca de imagens similares à uma imagem modelo (DATTA et al., 2008). Para utilização dessa técnica, imagens da utilização do software são primeiramente capturadas em uma versão funcional do software, para que sejam usadas como modelos. Após modificações no software novas imagens são novamente capturadas para que as propriedades extraídas das imagens (cor, textura, forma, etc.) sejam comparadas às propriedades extraídas das imagens modelos, com o objetivo de identificar falhas no software em teste.

- **Clustering -** Essa técnica utiliza algoritmos de agrupamentos para identificação de falhas em softwares. Para utilização desse método, dados de entradas, saídas e, opcionalmente, logs de execução de testes de uma aplicação são passados como parâmetros para algoritmos de agrupamento que serão responsáveis por separar grupos de testes executados corretamente e grupos de testes executados com falhas no software. Alguns exemplos de algoritmos de agrupamento utilizados nessa técnica são: *Agglometative hierarchical clustering* e *Expectation-maximization* (ALMAGHAIRBE; ROPER, 2016).

- **Pattern Mining -** Esse método tem por objetivo identificar falhas em softwares baseado em algoritmos de mineração de padrões frequentes. Basicamente, as execuções dos testes são gravadas como árvores de chamadas de funções, os algoritmos de mineração de padrões frequentes são utilizados para identificar subárvores frequentes em execuções de testes bem sucedidas e execuções com falhas.

- **Classification Tree -** Esse método consiste na construção de árvores de classificação para identificação de falhas em softwares e é baseado em especificações funcionais do software em teste, ou seja, requer que um modelo seja passado para construção das árvores de classificação. Assim, todos os fatores de influência (entradas, parâmetros, etc.) de uma aplicação são especificados usando uma representação gráfica.

- **DOM-Based -** Esse método é utilizado para identificação de falhas em aplicações web e é baseado na interação entre o DOM (*Data Object Model*) e o código JavaScript das aplicações. Tem por objetivo gerar automaticamente *assertions* usadas em nível de testes de unidades para código JavaScript das aplicações.

- **Reverse Engineering -** Essa abordagem tem por objetivo a identificação de falhas em software baseada em uma engenharia reversa do código-fonte de aplicações. Se-

gundo Binder (2000) os resultados esperados para os casos testes não necessariamente precisam ser conhecidos pelos testadores, são inferidos pelo código do SUT.

- **Mutant-based -** Essa técnica tem por objetivo automatizar a identificação de falhas no softwares por meio da utilização de versões mutantes SUT (com falhas inseridas no software). As informações do oráculo são obtidas de versões funcionais dos softwares (sem falhas) e comparadas às informações obtidas das versões mutantes durante a execução de testes para identificação de possíveis problemas nas aplicações.
- **Anomaly Detection -** Esse método realiza a detecção de comportamento inesperado em testes automáticos implementados pelos desenvolvedores e é utilizado em testes em nível de unidade. Realiza a identificação de falhas nos softwares baseando-se no grau de similaridade entre execuções de regulares e anormais.
- **Cookie Collection -** Abordagem utilizada para realização de testes em aplicações web, essa técnica automatiza o mecanismo de oráculo de testes baseando-se na captura de cookies de aplicações para verificação das modificações identificadas em coleções de cookies distintas obtidas de uma aplicação em teste.

A Tabela 5 apresenta o resultado da classificação dos trabalhos, indicando o total e quais trabalhos foram classificados em cada uma das técnicas identificadas nesse MSE. Com base na análise da tabela pode-se perceber que as principais técnicas identificadas na literatura para automação em oráculos de testes são: Model-based (37,5% dos trabalhos), Machine-learning (23,8% dos trabalhos) e Log-based (20,4% dos trabalhos).

A Figura 9 apresenta os resultados da classificação dos estudos primários quanto à **Técnica** utilizada para automação do oráculo (Eixo X) e **Contribuição** (Eixo Y).

3.1.7.4 SQ2 - Tipo de oráculo

A subquestão de pesquisa **SQ2** tem por objetivo classificar os diferentes tipos de oráculos de testes encontrados na literatura, levando-se em consideração os artefatos do SUT utilizados para automação dos oráculos de testes. Essa classificação foi proposta por Barr et al. (2015) e adaptada para esse MSE, pois este trabalho foca apenas em oráculos automáticos. Dessa forma, os trabalhos selecionados foram classificados em: *Specified Oracles, Derived Oracles* ou *Implicit Oracles*, conforme descritas na Seção 3.1.5.

Os resultados mostram que somente 6%(5) dos trabalhos foram classificados como *Implicit Oracles*, outros 36%(32) foram classificados como *Specified Oracles* e 58%(51) classificados como *Derived Oracles*. Dessa forma, pode-se perceber que grande maioria dos trabalhos que propõem oráculos automáticos, realizam a atividade por meio da utilização de artefatos que contém informações do SUT, como, especificações, diagramas, dados históricos do SUT, logs de execução, dentre outros.

Tabela 5 – Resultados da classificação dos trabalhos por Técnica de Automação.

Técnica de Automação	**Total (%)**	**ID's**
Model-based	33(37,5%)	[s05, s06, s07, s11, s15, s17, s19, s24, s26, s29, s32, s35, s37, s45, s46, s47, s52, s53, s58, s59, s61, s63, s65, s67, s68, s72, s76, s77, s79, s81, s83, s84, s87]
Machine-learning	21(23,8%)	[s08, s09, s12, s28, s30, s34, s35, s40, s42, s44, s49, s50, s51, s54, s57, s60, s64, s70, s73, s82, s85]
Log-based	18(20,4%)	[s04, s10, s21, s23, s27, s38, s39, s43, s55, s56, s62, s66, s69, s74, s75, s78, s80, s88]
GrO-method	3(3,4%)	[s22, s25, s36]
Clustering	2(2,2%)	[s01, s18]
Pattern Mining	1(1,1%)	[s71]
Classification Tree	1(1,1%)	[s02]
DOM-based	1(1,1%)	[s03]
Reverse Engineering	1(1,1%)	[s13]
Mutant-based	5(5,6%)	[s14, s20, s31, s41, s48]
Anomaly Detection	1(1,1%)	[s16]
Cookie Collection	1(1,1%)	[s86]

3.1.7.5 SQ3 - Tipo de Teste de Software

A subquestão de pesquisa **SQ3** foi elaborada com o intuito de identificar quais os tipos de testes de softwares podem ser realizados com apoio de oráculos de testes automáticos. Com base nos trabalhos analisados foi possível identificar diversos tipos de testes, são eles: Testes Funcionais, Testes de Regressão, Testes de Segurança, Testes de Desempenho e Testes de Mutação.

- **Testes Funcionais -** testes realizados com o intuito de encontrar inconsistências no software em relação aos requisitos funcionais.

- **Testes de Regressão -** testes aplicados em versões mais recentes de um software para garantir que após a inclusão de novas funcionalidades não surgiram defeitos em componentes já analisados.

- **Testes de Segurança -** testes realizados em softwares com o intuito de garantir a segurança das informações armazenadas em um software, alguns dos requisitos principais que um software deve garantir são confidencialidade, integridade, disponibilidade e autenticidade.

- **Testes de Desempenho -** tipo de teste realizado para identificar tempo de resposta de uma aplicação, determinando sua escalabilidade considerando uma carga de

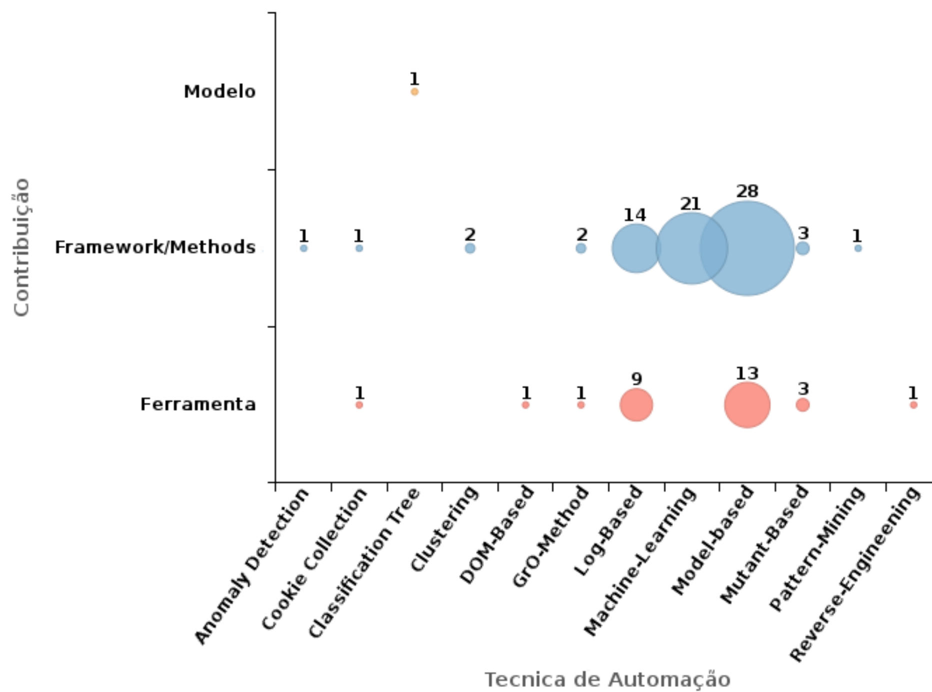


Figura 9 – Mapa Sistemático relacionando Contribuição e Técnica para automação.

acessos à aplicação, utilizado para identificar os gargalos de um sistema, dentre outras informações.

- **Testes de Mutação -** O teste de mutação requer a criação de diversos programas (mutantes) derivados a partir de um programa original relativamente correto. Segundo HOWDEN (1982), a definição de correção pode ser dada por: "Um programa P é correto em relação a uma função F se P computa F".

Analisando os resultados da classificação apresentados na Tabela 6 pode-se perceber que a maioria dos oráculos de testes automáticos (68% dos estudos analisados) oferecem suporte à realização de Testes Funcionais em softwares.

3.1.7.6 SQ4 - Restrições de Linguagem de Implementação

A subquestão **SQ4** foi elaborada com o intuito de identificar se os oráculos de testes automáticos disponíveis na literatura possuem restrições de uso em relação às linguagens de programação utilizadas na implementação dos SUT's. Assim, a visualização deste mapa pode auxiliar na escolha de um oráculo de testes adequado para cada software.

A Figura 10 apresenta os resultados da classificação dos trabalhos em relação às restrições de linguagem. Com base na análise da figura pode-se perceber que a maioria dos trabalhos encontrados nesse MSE propõem oráculos de testes sem dependência de utilização

Tabela 6 – Resultados da classificação dos trabalhos em relação ao Tipo de Teste.

Tipo de Teste	Total (%)	ID's
Testes Funcionais	60(68%)	[s01, s02, s04, s05, s07, s08, s09, s11, s13, s14, s15, s16, s17, s18, s24, s25, s26, s28, s29, s30, s32, s33, s34, s35, s37, s40, s43, s45, s46, s47, s48, s50, s52, s53, s57, s58, s59,s60, s61, s63, s64, s65, s66, s68, s69, s70, s71, s72, s73, s76, s77, s79, s81, s83. s84, s85, s86, s87, s88]
Testes de Regressão	23(27%)	[s03, s10, s12, s21, s22, s23, s27, s31, s36, s38, s39, s42, s44, s49, s55, s56, s62, s67, s74, s75, s78, s80, s82]
Testes de Segurança	1(1%)	[s06]
Testes de Desempenho	1(1%)	[s19]
Testes de Mutação	3(3%)	[s41, s51, s54]

de uma linguagem específica de implementação dos SUT's, entretanto, a utilização de alguns oráculos está condicionada a linguagem na qual o software foi implementado.

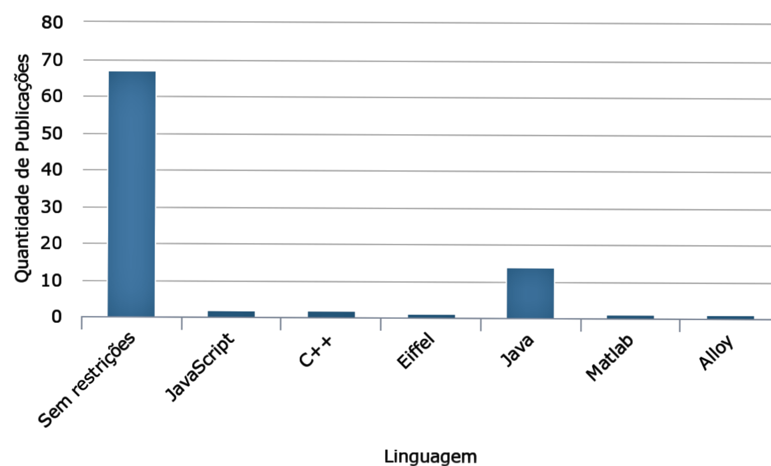


Figura 10 – Mapa Sistemático para Restrições de Linguagem.

3.1.7.7 SQ5 - Tipos de Softwares

Já subquestão **SQ5** tem por objetivo identificar se os oráculos de testes disponíveis na literatura possuem restrições de utilização em relação ao tipo se software implementado. Dessa forma, esse mapa auxilia os pesquisadores ou desenvolvedores na escolha de um oráculo de teste automático adequado ao software que está sendo implementado.

A Tabela 7 apresenta os resultados da classificação dos trabalhos em relação ao Tipo de Software, foram encontrados oráculos de testes específicos para diversos tipos, como: jogos, aplicativos para *smartphones* e TV's, aplicações para web, aplicações com interface gráfica de usuário (*GUI applications*), dentre outros.

Tabela 7 – Mapa Sistemático para Tipos de Softwares.

Tipo de Software	**Total (%)**	**ID's**
Não Específico	17(19%)	[s01, s09, s12, s14, s18, s28, s40, s42, s47, s49, s51, s54, s57, s64, s71, s73, s82]
Non-GUI applications	26(30%)	[s02, s13, s16, s19, s20, s26, s29, s30, s31, s33, s34, s35, s37, s41, s44, s45, s48, s50, s58, s59, s65, s67, s72, s74, s77, s87]
Data Acquisition Systems	1(1%)	[s32]
Distributed applications	2(2%)	[s38, s83]
GUI Applications	12(14%)	[s10, s25, s27, s36, s55, s68, s69, s78, s80, s81, s84, s88]
Games	1(1%)	[s11]
Mobile App's	3(3%)	[s23, s24, s53]
PDI applications	5(6%)	[s08, s36, s60, s70, s85]
Real-Time Systems	2(2%)	[s76, s79]
Embedded systems	1(1%)	[s63]
Web Applications	19(22%)	[s03, s04, s06, s15, s17, s21, s22, s25, s31, s36, s39, s43, s46, s56, s61, s62, s66, s75, s86]
Web Services	2(2%)	[s05, s07]
XML Processing Programs	1(1%)	[s52]

Com base na análise da Tabela 7 é possível observar que a maioria (30%) dos oráculos de testes encontrados são utilizados em softwares do tipo *Non-GUI applications*, ou seja, softwares que não possuem interface gráfica de usuário, ou ainda em processo de implementação, nos quais os oráculos são aplicados diretamente no código das aplicações. Além disso, destaca-se também o desenvolvimento de oráculos de testes voltados para aplicações acessadas por meio de um *browser* (*Web Applications*), foram encontrados 19 estudos que propõem oráculos de testes automáticos para esse tipo de software.

Em 17 estudos não foi possível identificar um tipo de software específico de software, principalmente, devido ao fato de existirem trabalhos que automatizam o oráculo de testes de forma independente do software que está sendo testado, ou seja, um conjunto de dados de execução do software é extraído utilizando um determinado método, seja baseado na documentação ou no conhecimento do domínio pelos desenvolvedores e são comparados com o mecanismo de oráculo de testes. Dessa forma, o oráculo de testes torna-se não específico para um determinado tipo de software. Apesar disso, existem *gaps* na área que podem ser abordados com novas pesquisas, pois as atividades relacionadas ainda dependem do conhecimento dos desenvolvedores, a automação desse processo ainda tem sido negligenciada.

Como principais lacunas em relação aos tipos de softwares, pode-se observar

que foram encontrados poucos trabalhos voltados para automação de oráculos de testes para: *Mobile App's* (3 trabalhos), *Games* (1 trabalho), *Real-Time Systems* (2 trabalhos), *Distributed Applications* (2 trabalhos) e outros.

É importante destacar que foram encontrados oráculos de testes que podem ser aplicados em mais de um tipo de software (s25, s31 e s36), os resultados da classificação individual dos trabalhos podem ser visualizados na Tabela 7.

3.1.7.8 SQ6 - Método de Avaliação Empírica

A subquestão **SQ6** foi elaborada com o objetivo de identificar quais métodos de avaliação empírica têm sido utilizados para avaliar os trabalhos que automatizam oráculos de testes, e assim, fornecer uma visão da área que facilite a escolha de um oráculo de testes com bons níveis de qualidade, baseando-se nas avaliações realizadas em cada trabalho. Segundo WOHLIN et al. (2012), na Engenharia de Software existem 3 métodos formais de avaliação empírica: *Survey*, Estudo de Caso e Estudo Experimental. Além disso, experimentos preliminares, realizados com o intuito de provar a eficácia do método proposto, foram classificados como Provas de Conceito, conforme descrito na Seção 3.1.5.

A Tabela 8 apresenta os resultados obtidos na classificação dos trabalhos em relação à subquestão **SQ7**, com base na sua análise foi possível identificar que a maioria dos trabalhos (52%) descreveram a avaliação do oráculo de teste proposto por meio de Provas de Conceito, cerca de 11% dos trabalhos não descreveram nenhuma avaliação. Tais resultados indicam uma carência de avaliações formais em pesquisas científicas relacionadas a automação em oráculos de testes, pois apenas 37% descreveram uma avaliação formal, seja por meio de Experimentos Controlados, Estudos de Caso ou Survey.

Tabela 8 – Mapa Sistemático para a questão Métodos de Avaliação Empírica.

Método de Avaliação Empírica	**Total (%)**	**ID's**
Estudo de Caso	19(21%)	[s03, s05, s07, s11, s12, s22, s27, s29, s32, s36, s37, s40, s43, s49, s55, s56, s61, s76, s86]
Experimento Controlado	14(16%)	[s01, s10, s14, s16, s19, s20, s21, s41, s60, s66, s69, s75, s78, s80]
Survey	0(0%)	-
Prova de Conceito	46(52%)	[s01, s06, s08, s09, s13, s15, s17, s18, s23, s24, s25, s26, s28, s33, s34, s35, s38, s42, s44, s47, s50, s51, s52, s53, s54, s57, s59, s62, s63, s64, s65, s67, s68, s70, s71, s72, s73, s74, s77, s81, s82, s83, s84, s85, s87, s88]
Não Avaliados	10(11%)	[s02, s04, s30, s31, s39, s45, s46, s48, s58, s79]

3.1.7.9 SQ7 - Tipo de software avaliado

Finalmente, a subquestão **SQ7** tem por objetivo identificar como os trabalhos que propõem oráculos de testes têm sido avaliados na literatura, baseando-se nos softwares que foram utilizados como objetos durante a avaliação do trabalho. Essa subquestão fornece um mapa que apoia os pesquisadores na escolha de um oráculo de testes com melhores níveis de confiança, dependendo da avaliação realizada, por exemplo, um oráculo de testes avaliado com "Softwares Reais" pode apresentar indícios de que tal método tenha um nível de confiança maior que um oráculo de testes avaliado com "Softwares Fictícios". No contexto dessa análise, foram considerados como experimentos os estudos de casos, *quasi*-experimentos, experimentos controlados, provas de conceito e exemplos de uso.

A subquestão **SQ7** foi proposta por OLIVEIRA; KANEWALA; NARDI (2015), e foi adaptada para utilização neste trabalho. Os tipos de softwares avaliados podem sem classificados em *Real Softwares* e *Toy Programs*, conforme descrito na Sub-seção 3.1.5.

Os resultados indicam que nos últimos anos o número de trabalhos avaliados por meio de experimentos em softwares reais tem aumentado, conforme pode ser observado na Figura 11. Além disso, é possível observar que 49% (45) dos trabalhos realizaram experimentos com softwares reais, 40% (36) realizaram experimentos com *toy programs* e apenas 11% (10) dos trabalhos não descreveram experimentos no conteúdo do trabalho. É importante destacar que alguns trabalhos realizaram experimentos com softwares reais e *toy programs*, são eles: s12, s15 e s17.

3.1.8 Discussão

Os resultados apresentados mostram uma tendência em um aumento de pesquisas publicadas referentes a automação de oráculos de testes, desde o ano de 2000, no que diz respeito aos principais eventos relacionados à Engenharia de Software. Essa tendência é consistente pois verificou-se que os trabalhos têm sido publicados em *Journals*, *Workshops*, Conferências e Simpósios altamente conceituados em diversos países, indicando um forte interesse pelo tema no campo da Engenharia de Software.

Dentro da área relacionada à automação em oráculos de testes, uma análise comparativa entre os trabalhos identificados por esse MSE pode ser realizada com o intuito de agregar, unificar e evoluir os oráculos de testes já existentes. Essa pesquisa pode ser útil para o progresso da área. Entretanto, quando há necessidade de propor um novo trabalho para determinada área, é importante que seja feita uma análise prévia da área para evitar que seja uma reinvenção de uma pesquisa já realizada e assim contribuir para a evolução e maturação da área. Dessa forma, destaca-se a importância desse MSE, pois pode ser utilizado por pesquisadores na realização de novas pesquisas relacionadas à automação em oráculos de testes.

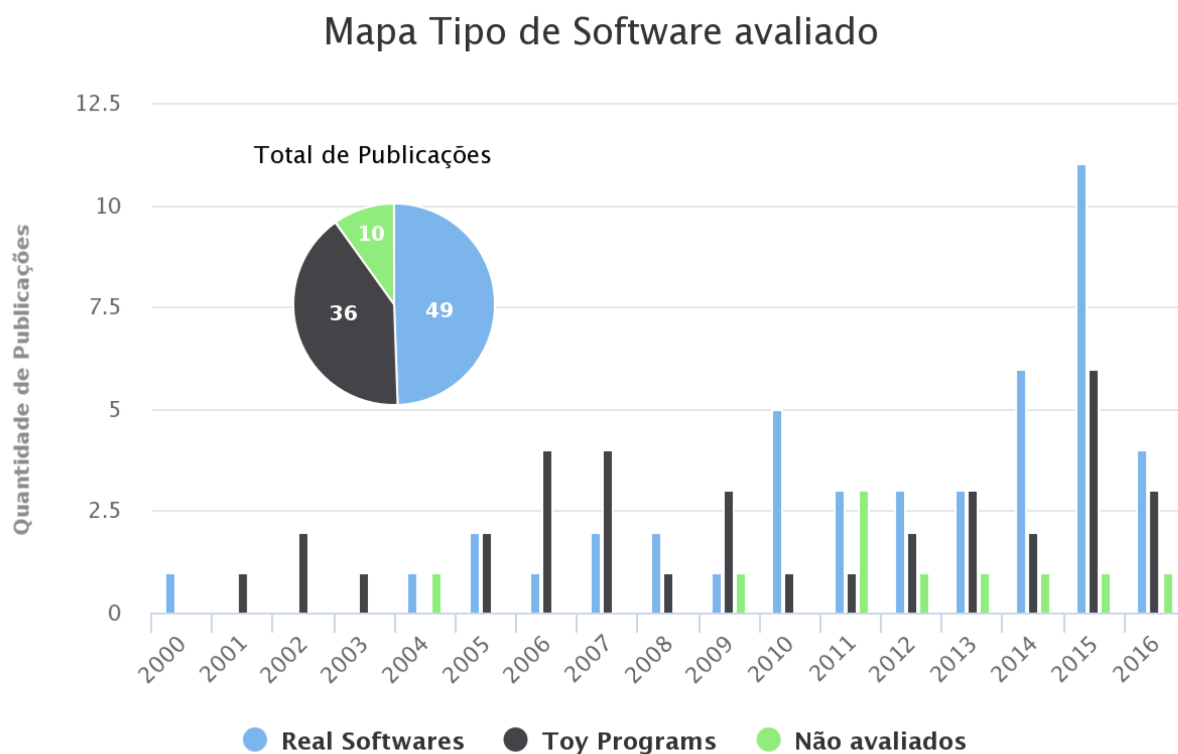


Figura 11 – Mapa Sistemático dos Tipos de Softwares avaliados.

Na literatura foram encontradas diversas técnicas que podem ser utilizadas na construção de um oráculo de testes automático. Dentre elas destacam-se técnicas baseadas em aprendizagem de máquina (**Machine Learning**), pois existem 21 trabalhos que realizam a automação de oráculos de testes com auxilio de uma ou varias técnicas dessa natureza, destacando-se como principais algoritmos: RNA (Redes Neurais Artificiais), SVM (do Inglês, *Support Vector Machine*), IFN (do Inglês, *Info-Fuzzy Networks*).

Outra técnica bastante utilizada na literatura é baseada na captura de logs de uma aplicação (**Log-based**). Com a utilização dessa técnica, os pesquisadores realizam a captura dos logs referentes aos eventos realizados em uma determinada aplicação, e criam um grafo mantendo um conjunto de nós (propriedades da aplicação) atingidos após a realização de cada um dos eventos (arestas do grafo). Esse tipo de oráculo de testes têm sido utilizado em Testes de Regressão, pois é necessário que exista uma versão funcional da aplicação para que seja realizada a captura de logs. Nesse MSE foram identificados 18 trabalhos que utilizam essa técnica para automação do oráculo de testes.

As técnicas baseadas em modelo (**Model-based**) também vêm sendo bastante explorada na literatura. Essa técnica consiste na utilização de artefatos modelos para identificação do domínio de entradas e saídas de uma determinada aplicação. Nesse tipo de oráculo, as saídas esperadas de um software são obtidas a partir de documentos do software, como, especificações, diagramas, modelos de execução, etc. Após da obtenção

do conjunto de saídas esperadas para os casos de testes, elas são comparadas com as saídas obtidas a partir da execução dos casos de testes. Uma dificuldade de utilização dessa técnica está relacionada a utilização de documentos dos softwares, pois, na prática, muitas empresas acabam negligenciando essa atividade por conta do esforço para criação de uma documentação. Neste MSE, foram identificados 33 trabalhos que automatizam o mecanismo de oráculo de testes utilizando a técnica **Model-based**.

Além de técnicas baseadas em aprendizagem de máquina, captura de logs e em modelos, foram identificadas na literatura diversas outras técnicas para automação de oráculos de testes, por exemplo, GrO-method (do Inglês, *Graphical Oracle Method*) é uma técnica que propõe um oráculo de testes baseado nas informações contidas em imagens extraídas da aplicação em teste, esse tipo de oráculo utiliza uma técnica chamada CBIR (do Inglês, *Content-Based Image Retrieval*) (DELAMARO; NUNES; OLIVEIRA, 2013) e tem sido bastante utilizado em aplicações destinadas a solução de problemas de visão computacional ou processamento de imagens. Também foram identificadas técnicas baseadas em: captura de *cookies* (TAPPENDEN; MILLER, 2014), engenharia reversa (ARANTES; SANTIAGO; VIJAYKUMAR, 2015), mineração de padrões (FATTA; LEUE; STEGANTOVA, 2006), técnicas de agrupamento (ALMAGHAIRBE; ROPER, 2015; ALMAGHAIRBE; ROPER, 2016), dentre outras.

Com a realização deste MSE também foi possível identificar como as técnicas para automação de oráculos de testes foram aplicadas na literatura, em quais tipos de softwares podem ser aplicadas, os tipos de testes de software suportados, como foram avaliadas, limitações de utilização dentre várias outras informações.

A Figura 12 apresenta o mapa da área relacionando as **Técnicas de Automação** com **Tipos de Software** nos quais foram aplicadas, assim como o mapa relacionando as **Técnicas de Automação** com os **Tipos de Testes** realizados. A partir da análise da figura é possível identificar as principais técnicas disponíveis na literatura utilizadas para apoio à realização de diferentes tipos de testes em vários tipos de software. Com base nessa análise é possível perceber que a técnica de automação baseada em modelo (Model-based) foi utilizada para automação do oráculo de testes na maioria dos tipos de softwares identificados neste MSE, entretanto, destaca-se a sua utilização em softwares sem interface gráfica de usuário (Non-GUI Applications), com 13 trabalhos identificados. Além disso, essa técnica foi utilizada para apoio à realização de diferentes tipos de testes de software, destacando-se sua utilização na realização de Testes Funcionais.

Ainda com base na análise da Figura 12, foi possível identificar outras Técnicas de Automação bastante utilizadas na literatura, sendo elas: *Machine-learning* e *Log-based*. Os oráculos de testes automáticos baseados em *Machine-learning* foram utilizados em alguns Tipos de Sofwares, como, *PDI Applications* e *Non-GUI Applications*, entretanto, a maioria dos trabalhos (12 estudos) utilizam o método em tipos de softwares não específicos. Esse

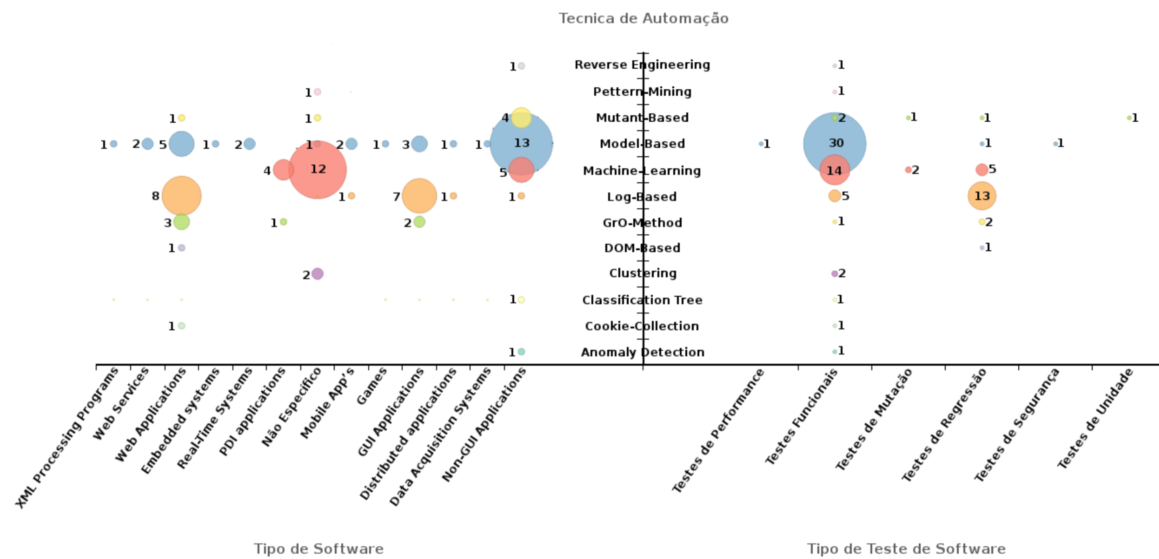


Figura 12 – Mapa Sistemático que relaciona Técnica de Automação com Tipo de Teste de Software e Técnica de Automação com Tipos de Softwares.

fato ocorre porque a maioria dos trabalhos utiliza a técnica de forma externa ao SUT, ou seja, um conjunto de informações relacionadas aos testes deve ser extraída dos SUT's para criação de uma base de dados de testes, que é utilizada para aplicação de algoritmos baseados em aprendizagem de máquina. Já a técnica *Log-based* tem sido bastante utilizada na literatura na automação dos oráculos de testes, principalmente, em softwares dos tipos *Web Applications* e *GUI-Applications*. Isso ocorre porque a técnica utiliza dados referentes aos logs capturados de aplicações. Esses logs, na maioria das vezes, são um conjunto de informações relacionadas à ações realizadas por usuários interagindo com uma interface gráfica das aplicações em teste, seja por meio de um *browser* ou uma janela da própria aplicação.

3.1.9 Ameaças à Validade

Existem alguns fatores que podem ameaçar à validade de um MSE. Foram identificados alguns desses fatores com o objetivo de mitigar suas influências nos resultados deste estudo.

- **Cobertura da string de busca -** Essa ameaça refere-se à eficácia da string utilizada para busca de trabalhos relevantes à área. Para minimizar o efeito dessa ameaça foi utilizada uma string construída de forma mais abrangente possível, para isso, foram utilizados sinônimos dos diferentes termos. Além disso, foi realizada uma pesquisa piloto em diferentes bases de dados para verificar a consistência da string utilizada.

- **Seleção de Estudos -** Outro fator que pode ameaçar à validade de um MSE está relacionado ao processo de seleção de trabalhos relevantes à área pesquisada. Essas etapas podem ser enviesadas pela opinião dos pesquisadores que estão executando o processo. Para mitigar essa ameaça, a seleção de estudos foi realizada por diferentes pesquisadores de maneira independente, e foram realizadas análises de validade após a realização de cada etapa de seleção, para verificar a concordância entre os pesquisadores, conforme descrito na Seção 3.1.4.

- **Extração de Dados -** Essa ameaça refere-se à possibilidade de diferentes pesquisadores interpretarem as informações dos trabalhos de maneiras diferentes. A primeira ação realizada para mitigar essa ameaça foi a utilização de planilhas eletrônicas para facilitar a organização das informações extraídas referentes a cada questão de pesquisa. Dessa forma, a utilização de planilhas permitiu uma melhor comparação das informações extraídas entre os pesquisadores. Os trabalhos em que houve discordâncias de informações extraídas foram novamente analisados e discutidos até que houvesse consenso entre os pesquisadores.

3.2 Oráculos de Testes não incluídos no MSE

Na literatura foram identificadas outras técnicas que podem ser utilizadas para apoio à realização de testes em softwares quando a utilização de um mecanismo de oráculo de testes automático é inviável. Esses trabalhos não foram incluídos no MSE, pois o objetivo do mapeamento foi definido para a identificação de trabalhos que visem propor oráculos automáticos. No entanto, é importante destacar os principais conceitos relacionados à essas técnicas.

3.2.1 Pseudo-Oráculos

Esse conceito foi introduzido por Davis e Weyuker (1981), que tinham a proposta de utilização de "falsos oráculos" em programas "não-testáveis". Um pseudo-oráculo corresponde à uma versão independente da aplicação desenvolvida, por exemplo, por equipes diferentes, em diversas linguagens de programação. Essas diferentes versões do software, geralmente, são executadas em paralelo durante a realização de testes com o objetivo de comparar resultados entre as aplicações.

Os Pseudo-oráculos podem ser utilizados como uma alternativa quando a utilização de um oráculo automático for inviável, entretanto, essa técnica apresenta algumas desvantagens, por exemplo, o esforço e custo de implementação será dobrado, visto que é necessário construir duas versões diferentes da aplicação. Além disso, existem dificuldades na identificação de problemas no software, pois mesmo que o resultado da execução de um

teste seja diferente nas duas versões, não há garantias que ambas não possuam erros de implementação.

3.2.2 Testes Metamórficos

Essa técnica tem sido utilizada na literatura para realização de testes em aplicações complexas (DING; ZHANG, 2016; CHAN; CHEUNG; LEUNG, 2005), nas quais a identificação de resultados esperados para determinados casos de testes é inviável, e assim o oráculo de testes convencional é impraticável. Nesse caso, a verificação de testes nas aplicações está relacionada a identificação de Relações Metamórficas (propriedades da aplicação) que devem ser satisfeitas durante a realização de testes. Por exemplo, uma aplicação responsável pelo cálculo do seno de um ângulo é difícil de ser testada, pela dificuldade em identificar o resultado esperado exato para o seno de um ângulo, no entanto, uma relação metamórfica identificada para a aplicação é: *seno(x) = seno(180 - x)*. Essa propriedade deve ser mantida durante a execução de testes e assim é possível verificar a corretude da aplicação.

Segundo Oliveira, Kanewala e Nardi (2015), a aplicação de testes metamórficos pode ser realizada em quatro etapas básicas, são elas:

1) Identificar as relações metamórficas que devem ser satisfeitas pelo SUT;

2) Criar um conjunto inicial de casos de testes, que pode ser realizada utilizando abordagens tradicionais como testes aleatórios ou testes baseados em falhas;

3) Criar um conjunto de casos de testes utilizando as relações metamórficas criadas na etapa 1;

4) Executar os conjuntos de casos de testes criados nas etapa 2 e 3 e verificar se as mudanças nas saídas ocorreram de acordo com as mudanças de saídas especificadas pelas relações metamórficas.

Um dos maiores desafios na automação de testes de software utilizando a técnica de testes metamórficos está relacionado à identificação das relações metamórficas de forma automática, pois cada relação é extraída de acordo com o contexto para o qual o software foi projetado. Geralmente, essas relações são extraídas baseadas no conhecimento do domínio pelos próprios desenvolvedores. Por esse motivo essa técnica não foi incluída no escopo do MSE. No entanto, esse fato não minimiza os impactos da utilização de testes metamórficos para apoio à automação em testes de softwares.

3.3 Principais Trabalhos Relacionados

Este capítulo discutiu 88 trabalhos identificados no Mapeamento Sistemática que visam automatizar oráculos de testes em softwares. No entanto, apesar dos diversos trabalhos relacionados à essa linha de pesquisa, alguns são destacáveis por ter foco em oráculos de testes para aplicações web, que é o foco da experimentação neste trabalho. Essa seção destaca alguns desses trabalhos descrevendo de forma sucinta o método proposto em cada um deles.

Mirshokraie, Mesbah e Pattabiraman (2013) propõem um método para automação de oráculos de teste para aplicações *JavaScript* que opera em três passos: infere um modelo de teste; gera casos de teste em níveis de unidade para funções *JavaScript*; e usando testes de mutação gera automaticamente os oráculos de teste. No primeiro passo, é explorado dinamicamente vários estados de uma determinada aplicação web, de maneira a maximizar o numero de funções cobertas durante a execução, gerando um grafo de fluxo de estados, capturando os estados DOM dinâmicos explorados e as transições baseadas em eventos entre eles.

Em (ROEST; MESBAH; DEURSEN, 2010) é proposta uma técnica para automação de oráculos de teste, para testes de regressão, em aplicações web. Tal método utiliza, como técnica de automação, uma estratégia de captura de logs de execução do sistema (*Log-based*). Nessa abordagem, cada comparador é especializado em comparar as diferenças entre as árvores DOM (*Document Object Model*), inicialmente separando as diferenças definidas por meio de uma expressão regular ou uma expressão *XPath*, para em seguida fazer a comparação. A aplicação desses comparadores torna-se muito eficaz ao ignorar diferenças não deterministas em tempo real, mas podendo sofrer com erros reais por dados perdidos.

Delamaro, Nunes e Oliveira (2013) propõem um *framework* que pode ser usado para criar oráculos para programas em diversas plataformas. No trabalho é apresentado um método alternativo para a construção e automação de oráculos de teste, usando os conhecidos conceitos de CBIR (*Content-Based Image Retrieval*). Tal método, chamado de *GrO-method*, utiliza como informação de oráculo informações extraídas de arquivos de imagens obtidas da aplicação em teste e é aplicado em oráculos para testes de regressão, sendo avaliado primeiramente em sistemas CAD (sistemas computacionais acoplados a equipamentos médicos especializados, para dar assistência em diagnósticos), mas avaliado também em aplicações Web.

Já em (SPRENKLE et al., 2005) é proposto um *framework* para aplicações web baseado na captura de ações executadas na aplicação (*Log-based*), gerando novas seções de usuário com o objetivo de replicação de seções para comparação de requisições e respostas do servidor. A estrutura projetada para o *framework*, coleciona seções de usuários, realiza

testes com as aplicações web e avalia a eficácia dos casos de teste com o mínimo de esforço humano. No trabalho ainda foram apresentadas duas técnicas de repetição e vários comparadores de oráculos no contexto de uma estrutura automatizada para testar aplicações web.

Em (JAHANGIROVA et al., 2018) é proposta a ferramenta OASIs, uma ferramenta baseada em busca para automação do oráculo de teste em aplicações JAVA. Utilizam técnicas de geração de casos de testes e testes de mutação para a identificação de falhas nas aplicações. A ferramenta busca realizar a análise de duas propriedades em invariantes do programa: a **Completude**, que analisa se os estados corretos do software são aceitos pelo oráculo. A **Solidez**, que verifica se os estados defeituosos são rejeitados pelo oráculo. A OASIs é uma ferramenta baseada em linhas de comando que recebe como entrada alguns parâmetros indicando a parte do código-fonte do projeto que será analisada e devolve como resultado um relatório indicando as deficiências encontradas.

Em (KIRAÇ; AKTEMUR; SÖZER, 2018) é proposta uma ferramenta denominada VISOR, que tem por objetivo a automação do oráculo de testes para sistemas de saída visual. A ideia da proposta é determinar a corretude dos sistemas com base na comparação de imagens obtidas como saídas do sistema. O trabalho tem com foco a abordagem de testes caixa-preta em equipamentos eletrônicos. Os casos de testes são avaliados comparando *snapshots* com imagens referências previamente capturadas das aplicações. Avaliaram o trabalho em um estudo de caso aplicado na realização de testes de regressão em um sistema comercial de TV digital.

Cada um dos trabalhos apresentados tem suas particularidades. São diversos métodos encontrados na literatura que propõem diferentes maneiras para automação de oráculos de testes em aplicações web, e com um foco específico, por exemplo, alguns são voltados para testes de regressão, outros geram oráculos de testes em níveis de unidade. O método proposto nesta dissertação não visa substituir técnicas propostas anteriormente, mas sim uma nova alternativa nessa linha de pesquisa, diferenciando-se dos demais trabalhos em relação à forma como o oráculo é proposto, uma vez que neste trabalho a abordagem foi projetada com o intuito de ser aplicada para testes de software tanto em ambiente de produção quanto em ambientes de homologação. Além disso, foi investigado o uso de outras técnicas de aprendizagem de máquina ainda não exploradas na literatura para a solução do problema do oráculo de teste.

3.4 Considerações Finais

Este capítulo apresentou um Mapeamento Sistemático realizado com o objetivo de identificar, analisar e sumarizar a literatura disponível relacionada à automação em oráculos de testes. Esse estudo fornece uma visão geral dessa linha de pesquisa, identificando os

principais tópicos abordados por trabalhos anteriores e destacando as principais lacunas passíveis de investigação. A realização desse mapeamento ajudou a direcionar a pesquisa sobre a automação do oráculo de teste, pois foi possível perceber que não existiam trabalhos que combinassem a utilização de informações de *logs* da aplicação com técnicas de aprendizagem de máquina. Além disso, não foi identificado nenhum trabalho na literatura que investigasse o uso de técnicas de *Boosting* para auxiliar na resolução do problema. Isso atende ao segundo objetivo desta dissertação (Seção I).

Parte III

Proposta

4 Abordagem Proposta

Com o objetivo de reduzir custos e complexidade envolvidos na realização de testes, este trabalho propõe uma abordagem para automação de parte dessa atividade: a geração automática de oráculos de testes. O método proposto é baseado na captura de *logs* de utilização e análise das informações com auxílio de técnicas de aprendizagem de máquina. Neste trabalho, abordagem proposta foi utilizada para auxiliar a realização de Testes de Regressão em aplicações Web. Portanto, é apresentado um exemplo da implementação do método para aplicações Web.

4.1 O Método

O método proposto neste trabalho é baseado na captura de ações realizadas por usuários de aplicações e na utilização de uma técnica de aprendizagem de máquina para automação do oráculo de testes. A ideia principal é comparar as informações capturadas da aplicação com a informação do oráculo referente a mesma ação, visando identificar eventuais inconsistências na aplicação.

Para que seja possível a geração do oráculo, o método proposto aqui possui quatro etapas: Captura de Ações, Preparação de Dados, Aprendizagem de Máquina e Avaliação da Aprendizagem. A Figura 13 apresenta as etapas do método proposto.

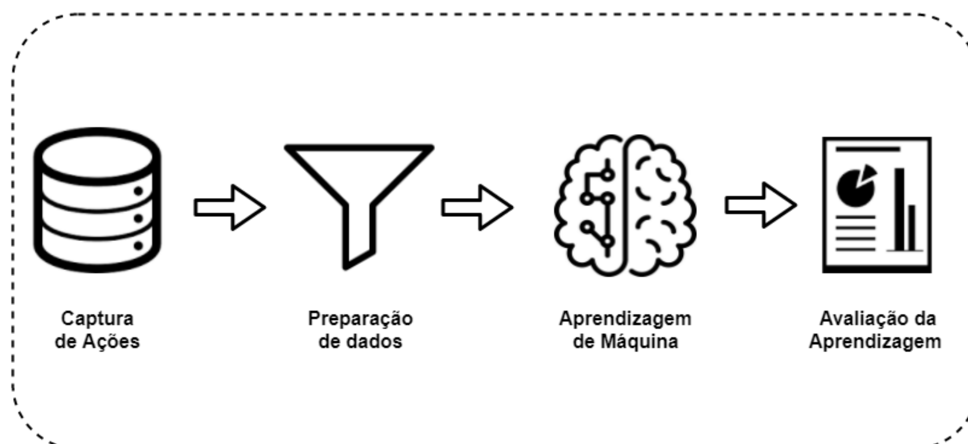


Figura 13 – Abordagem proposta para automação do oráculo de teste.

Na etapa de Captura de Ações, deve ser inserido no SUT um mecanismo para realização de captura de ações executadas na aplicação e envio para um servidor responsável pelo armazenamento de tais informações. Na etapa de preparação de dados é realizado um processamento na base de dados que será utilizada no algoritmo de aprendizagem de máquina, sendo realizado, por exemplo, remoção de informações desnecessárias e inclusão

de ações associadas a um funcionamento indevido do software, visando com isso introduzir eventos associados ao funcionamento correto e incorreto do SUT. Durante a etapa de Aprendizagem de Máquina é realizado o treinamento e teste com os dados obtidos nas etapas 1 e 2. Finalmente, na Etapa 4 são calculadas um conjunto de medidas que detalham o desempenho da técnica de aprendizagem de máquina para automação do oráculo de testes.

4.1.1 Captura de Ações

A etapa de Captura de ações é destinada à coleta de eventos realizados por usuários na aplicação. No método proposto neste trabalho é sugerida a realização da captura no lado do cliente, pois permite um detalhamento maior das ações que não seria possível no lado do servidor. Essa captura pode ser realizada por meio da inserção de um mecanismo de captura na aplicação em teste. A Figura 14 ilustra um diagrama exemplificando esse processo. Pode-se perceber que o após a inserção de um componente de captura na aplicação em teste, um conjunto de *logs* contendo as ações realizadas na aplicação é capturado e enviado à um servidor para armazenamento dessas informações e as requisições ao servidor da aplicação seguem seu fluxo normal.

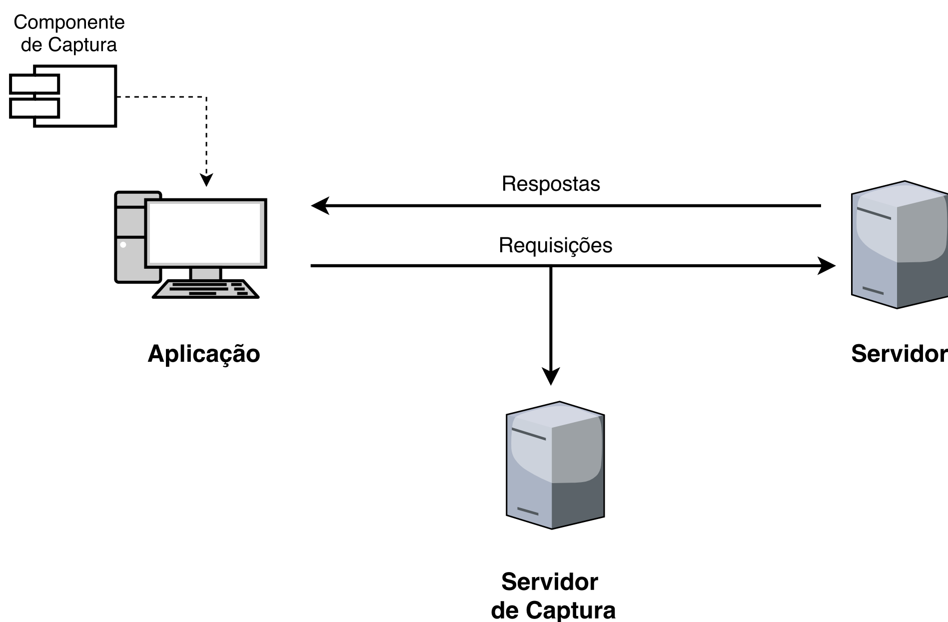


Figura 14 – Exemplo ilustrando a etapa de captura de ações.

Com a captura de ações realizadas no lado do cliente é possível obter um conjunto de informações que mapeiam todos os eventos realizados durante a utilização da aplicação, essas informações podem conter:

- Tempo: representando o momento em que a ação foi executada na aplicação, dessa forma é possível obter uma ordem cronológica de ações;

- Elemento: corresponde ao objeto que sofreu a ação na aplicação, por exemplo, um botão, um campo de texto, etc;
- Quem fez a ação: é possível coletar qual usuário realizou a ação;
- Tipo de Ação: indica qual ação foi realizada, como: cliques, preenchimento, mouse sobre e outras;
- Onde a ação ocorreu: representa a interface da aplicação em que a ação foi realizada, por exemplo, uma tela, uma página web, etc;
- Conteúdo: essa informação representa o conteúdo do elemento que sofreu a ação, pode ser: o texto preenchido em um campo de formulário, o título presente em um botão, etc;

Neste trabalho, durante a etapa de captura de ações também são capturadas informações que irão compor as saídas esperadas da aplicação para cada ação capturada. Existem diversas informações que podem ser utilizadas como saídas esperadas, por exemplo, número de elementos visíveis em uma tela após a realização de uma ação, título da tela a ser carregada após a execução da ação, imagem de uma interface da aplicação, mensagens que serão exibidas após a realização de uma ação e outras.

4.1.2 Preparação de Dados

A etapa de preparação de dados é composta por um conjunto de atividades que visa preparar a base de dados para utilização na técnica de aprendizagem de máquina. Essa etapa é realizada com o intuito modificar a base de dados que será utilizada no problema de forma a melhorar o desempenho do algoritmo de aprendizagem de máquina para generalização da solução em novos problemas. Algumas atividades que podem ser realizadas na preparação de dados são:

- Remoção de ruídos: existem alguns exemplos que podem estar presentes na base de dados de forma esporádica, ou seja, aquela informação pode prejudicar o desempenho do algoritmo sendo que esse dado pode dificilmente ocorrer novamente na aplicação;
- Remoção de informações irrelevantes: podem existir algumas informações capturadas que podem não ser relevantes para a solução do problema. Essas informações podem ser removidas sem prejuízos para o aprendizado do algoritmo de AM;
- Inserção de exemplos: neste trabalho as ações capturadas na etapa de captura de ações foram consideradas válidas, ou seja, sem erros na aplicação. No entanto, para que o algoritmo de AM identifique as falhas nas aplicações foi necessário inserir na base de dados um conjunto de exemplos que representam as falhas na aplicação.

Nesta etapa do método devem ser realizadas todas as operações necessárias na base de informações capturadas das aplicações para serem analisadas pelas técnicas de aprendizagem de máquina.

4.1.3 Aprendizagem de Máquina

Na etapa de aprendizagem de máquina, o conjunto final de ações é utilizado em um algoritmo de aprendizagem de máquina, para criação de um modelo que irá representar o mecanismo de oráculo de testes. O objetivo dessa etapa é criar um classificador capaz de identificar um comportamento do SUT que possa estar associado com uma situação de falha.

Existem diferentes técnicas de aprendizagem de máquina que podem ser utilizados para solução do problema. Por exemplo, Redes Neurais Artificiais, técnicas baseadas em árvores de decisão e outras. A escolha de uma técnica pode influenciar diretamente nos resultados do problema. Além disso, cada técnica pode ser configurada de diversas formas por meio de hiper-parâmetros de configurações. Essas configurações também podem impactar na resolução do problema do oráculo.

Por conta do exposto, essa etapa do método é destinada a escolha de uma técnica e de hiper-parâmetros de configurações que melhor se adequem à resolução do problema de oráculo de teste levando-se em consideração a métrica de taxa de erro.

4.1.4 Avaliação da Aprendizagem

A última etapa do método consiste na avaliação da técnica de aprendizagem de máquina utilizada como oráculo de testes. Neste trabalho, essa avaliação foi realizada por meio do cálculo de algumas medidas, comumente utilizadas na literatura (SOKOLOVA; LAPALME, 2009), para avaliações de classificadores. Os principais conceitos relacionados às medidas utilizadas foram apresentados no Capítulo 2.

4.2 Implementação do Método

Esta seção detalha a implementação do método proposto. Para isso, foi necessário definir as tecnologias a serem usadas para o contexto da implementação.

4.2.1 Implementação da Captura de Ações

Para que seja possível a realização de testes em aplicações, de forma remota e assíncrona, um conjunto de *logs* é capturado e enviado à um servidor responsável pelo armazenamento e processamento dessas informações. A captura dessas ações é realizada por meio de um componente de captura de eventos. Como neste trabalho foi utilizada uma

aplicação Web como objeto de experimentação, foi utilizado um arquivo em *JavaScript*, que interage com navegadores Web, para realizar tal operação. Esse componente é personalizável e deve ser inserido em cada uma das páginas ou *templates* da aplicação que será testada, dessa forma, as ações realizadas por usuários nas páginas nas quais o componente foi inserido serão enviadas ao servidor. Dentre as informações capturadas, algumas são obrigatórias pois são essenciais para utilização na etapa de aprendizagem de máquina, são elas:

- **Tempo:** o horário em que a ação ocorreu;
- **Aplicação:** nome para identificação da aplicação que a ação ocorreu;
- **Elemento que sofreu a ação:** link, campo de texto, botão, etc.;
- **Tipo da Ação:** clique, sobreposição do mouse, preenchimento de campo, duplo clique, carregamento de página, etc.;
- **Onde a ação ocorreu:** página na qual a ação foi realizada, utilizado URL da página para identificação;
- **Quem fez a ação:** identificador do usuário que realizou a ação;
- **Conteúdo:** essa informação representa o conteúdo do elemento que sofreu a ação, pode ser: o texto preenchido em um campo de formulário, o título presente em um botão, etc;
- **Número de elementos visíveis:** quantidade de elementos da página que são visíveis ao usuário após a execução da ação;
- **Página a ser carregada:** URL da página, que é carregada após a execução da ação;

Para este trabalho, dentre as informações capturadas em cada ação, os atributos **Número de elementos visíveis** e **Página a ser carregada** compõem a informação do oráculo das ações realizadas, ou seja, são os resultados esperados após a realização de uma determinada ação na aplicação.

É importante destacar que as informações capturadas não identificam exatamente o que o usuário da aplicação realizou, pois é importante manter a privacidade dos dados dos usuários e manter confidencialidade das ações realizadas. Por esse motivo, a captura de ações é realizada no lado cliente da aplicação por meio de um *script* configurável pelo usuário.

4.2.2 Implementação da Preparação de Dados

Para a preparação dos dados foram realizadas duas atividades principais. Inicialmente foram removidas informações irrelevantes das ações capturadas na etapa anterior. Essa remoção foi realizada com base na análise do ganho de informação de cada um dos atributos. Após a remoção dessas informações, foram inseridos, na base de dados, exemplos com falhas nos casos de teste. Essa medida foi adotada porque as ações que compõem a base de dados foram capturadas de uma versão funcional da aplicação, ou seja, foram consideradas que as informações capturadas estão corretas.

Para inserção das falhas, foi inicialmente extraído um subconjunto do total de ações capturadas na etapa de captura de ações. Esse subconjunto foi extraído de forma aleatória, tentando minimizar a escolha de ações tendenciosas a facilitar a identificação de falhas pelo algoritmo de aprendizagem máquina. Geralmente, esse subconjunto é gerado contendo em torno de 30% a 50% do total de ações.

As falhas são inseridas no subconjunto extraído por meio da modificação da informação do oráculo em cada uma das ações, ou seja, o número de elementos visíveis e página a ser carregada são modificadas de forma aleatória para geração do conjunto de dados com falha e duvidosos que serão utilizados na etapa de aprendizagem de máquina. Após as modificações realizadas, esse novo subconjunto é acrescentado ao conjunto original de ações, e esse novo conjunto foi utilizado para treinamento e testes do algoritmo de aprendizagem de máquina. A Figura 15 ilustra a atividade de inserção de falhas durante a etapa de preparação de dados.

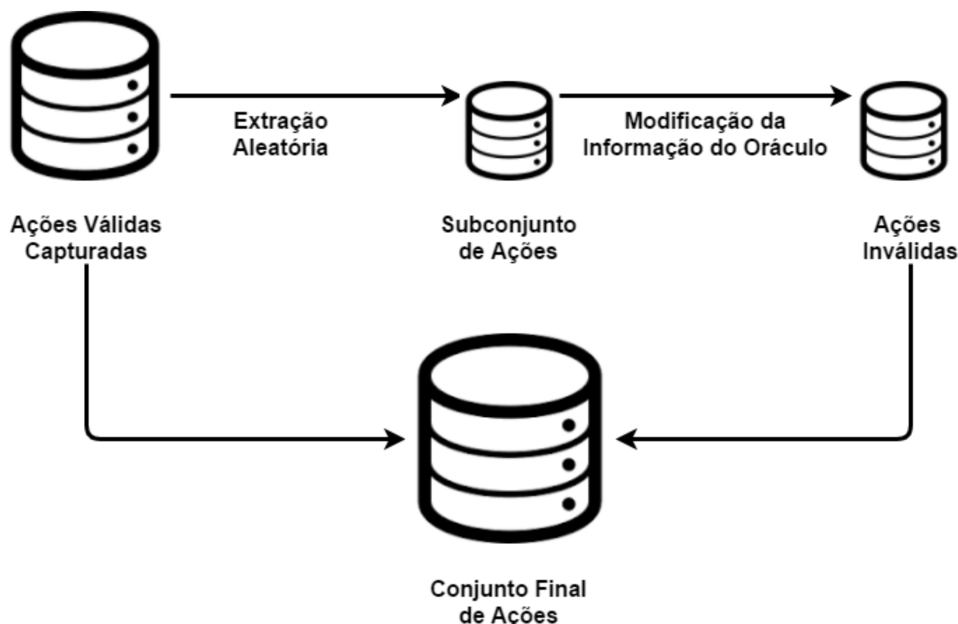


Figura 15 – Etapa de preparação de dados.

É importante destacar que a extração do subconjunto de ações deve ser realizada de forma aleatória com o intuito de mitigar ameaças à validade do método relacionadas a

uma seleção de ações tendenciosas. Além disso, as modificações na informação do oráculo também devem ser realizadas de forma aleatória, mas dentro do domínio de valores possíveis para os atributos que compõem a informação do oráculo (Número de elementos visíveis e URL do oráculo, neste trabalho).

4.2.3 Implementação da Aprendizagem de máquina

Neste trabalho, a etapa de aprendizagem de máquina foi iniciada pela definição da topologia em relação às entradas e saídas do algoritmo de AM. Após a definição da topologia foi realizada a seleção de um algoritmo de AM e seus hiper-parâmentros. O algoritmo de aprendizagem de máquina irá classificar cada um dos testes a serem executados no SUT em uma das seguintes classes: **Válido**, **Falha** e **Possível Falha**.

Testes que tenham como classificação o rótulo *Válido* indicam que sua execução não possui indícios de falha. Testes classificados com rótulo **Falha** indicam foi detectada uma situação incorreta no SUT. Testes classificados como **Possível Falha** são aqueles em que as saídas esperadas divergem das saídas obtidas da aplicação, no entanto, essa diferença não é suficiente para que o classificador indique uma falha no funcionamento do SUT. Por exemplo, suponha que um caso de teste contenha os valores esperados de 123 para **Número de elementos visíveis** e "http://aplicacao.com/login" para a **Página a ser carregada**. Supondo que os valores obtidos como saídas da aplicação sejam 80 para **Número de elementos visíveis** e "http://aplicacao.com/login" para **Página a ser carregada**, esse caso de teste poderá ser classificado como "Possível falha", pois existe apenas uma diferença no número de elementos visíveis, não sendo possível afirmar que exista um problema na aplicação, uma vez que, após uma mesma ação executada em momentos diferentes, podem aparecer nas telas do SUT outros elementos não existentes em execuções anteriores. Como este trabalho faz parte de um projeto maior com o objetivo de aplicar o oráculo de teste automático para apoio à realização de testes de regressão contínuo em aplicações, optou-se por criar a classe "Possível Falha" que, em conjunto com a classe "Falha", compõem o conjunto de ações a serem investigadas pelos desenvolvedores das aplicações em teste.

A Tabela 9 apresenta a topologia do classificador utilizado neste trabalho em relação ao número de entradas e saídas, assim como seus respectivos domínios. É importante destacar que essa topologia foi criada baseada nos experimentos realizados neste trabalho, no entanto, é possível realizar modificações acrescentando novas entradas, podendo variar de acordo com a aplicação. Por exemplo, aplicações podem ter funcionalidades diferentes dependendo do perfil de usuário, assim, tal informação também pode se tornar necessária para correta classificação dos casos de testes apresentados ao classificador.

Tabela 9 – Entradas e saídas do classificador.

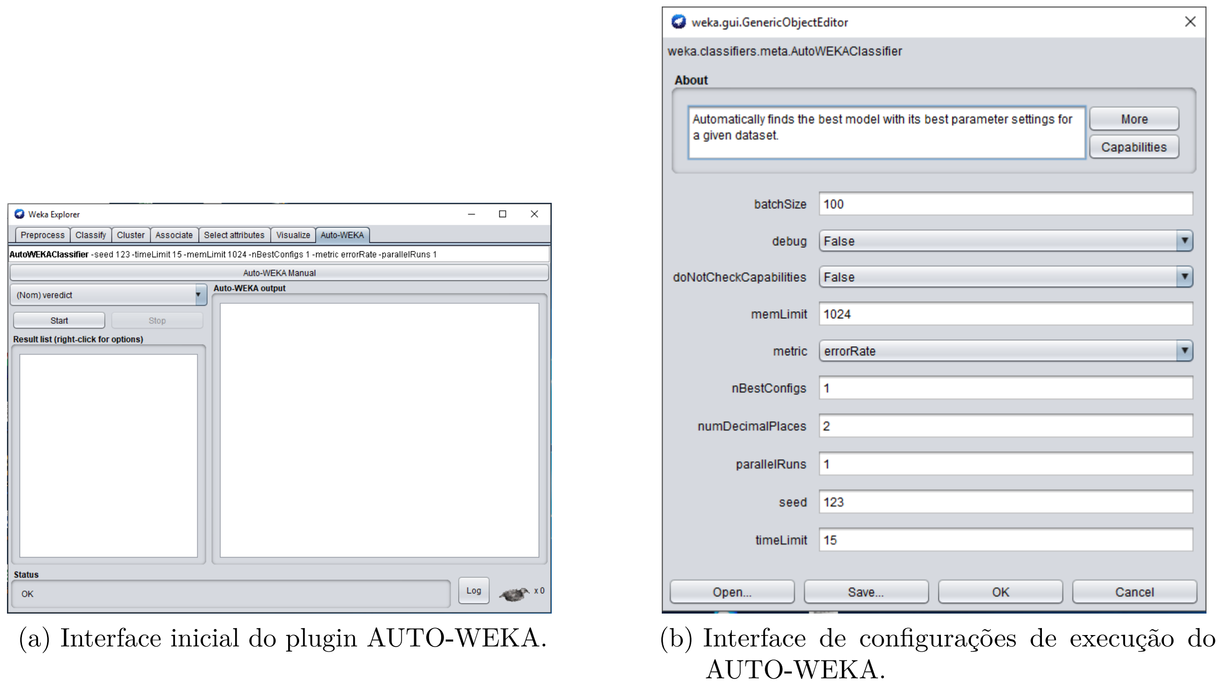
Tipo	Nome	Domínio
Entrada	sActionType	String
	sTag	String
	sTagIndex	Numérico
	sUrl	String
	sXPath	String
	sOracleVisibleElements	Numérico
	sOracleUrl	String
Saída	Veredito	{ Válido, Inválido, Possível falha }

4.2.3.1 Seleção de Classificador

Para este trabalho, optou-se por utilizar a ferramenta *Waikato Environment for Knowledge Analysis* (HALL et al., 2009), mais conhecida como WEKA, tem como objetivo fornecer um conjunto abrangente de algoritmos de aprendizado de máquina e ferramentas de pré-processamento de dados para pesquisadores e profissionais. A WEKA possui uma interface gráfica e permite que tal ambiente seja estendido de várias formas sem a necessidade de modificar seu núcleo. Além disso, possui um conjunto de relatórios que facilita a análise do desempenho dos algoritmos. Essas funcionalidades tornam o WEKA um dos software mais bem conhecidos para auxiliar na pesquisa envolvendo aprendizado de máquina e mineração de dados.

Para a escolha de um classificador que fosse melhor adaptado ao problema, foi utilizado um plugin da ferramenta WEKA denominado AUTO-WEKA (THORNTON, 2014). Esse plugin realiza a seleção de algoritmos e hiper-parâmetros para os algoritmos de classificação e regressão implementados na WEKA, ou seja, dado um específico conjunto de dados, o AUTO-WEKA utiliza diferentes configurações de hiper-parâmetros com diversos algoritmos e sugere o método que obteve melhor desempenho de generalização para o problema. A Figura 16a apresenta a interface gráfica de utilização do plugin AUTO-WEKA, e na Figura 16b é apresentada os parâmetros configuráveis pelo usuário de acordo com sua necessidade. Dentre eles é importante destacar o parâmetro *metric*, que representa a configuração da métrica que será utilizada para escolha do algoritmo e o parâmetro *timeLimit*, que representa o intervalo de execução no qual serão realizados testes para solução do problema. Esse valor é diretamente proporcional ao número de algoritmos e configurações testadas durante a execução do plugin.

Neste trabalho, a escolha do algoritmo foi baseada no desempenho do algoritmo durante os experimentos realizados. O algoritmo com melhor desempenho foi o *Adaptative Booting*, popularmente conhecido como *AdaBoost*. Conforme discutido na Seção 2.2, esse algoritmo trabalha em conjunto com um algoritmo base. O algoritmo selecionado com base nos experimentos realizados foi o JRIP, que é uma implementação do algoritmo IREP



(a) Interface inicial do plugin AUTO-WEKA.

(b) Interface de configurações de execução do AUTO-WEKA.

Figura 16 – Interface Gráfica da Ferramenta WEKA.

(Seção 2.1), na ferramenta WEKA. No Capítulo 2 foi apresentado os principais conceitos relacionados ao algoritmo utilizado neste trabalho. O detalhamento dos experimentos realizados será apresentado no Capítulo 5.

4.2.3.2 Preparação da Aprendizagem de Máquina

Como a abordagem proposta neste trabalho sugere a utilização técnicas de AM com aprendizado supervisionado, para que o algoritmo seja utilizado como oráculo de testes, é necessário realizar inicialmente o treinamento do algoritmo utilizando exemplos de treinamento compostos por entradas e seus correspondentes resultados esperados. O conjunto de exemplos utilizado como treinamento deve ser correto e conter o mínimo de ruídos possíveis para correta aprendizagem do algoritmo. Após a realização do processo de treinamento, o algoritmo deverá ser testado com um conjunto de exemplos diferentes do conjunto de treinamento utilizado para verificação do seu desempenho para generalização do problema.

Neste trabalho, os conjuntos de exemplos para treinamento e teste foram aleatoriamente escolhidos do conjunto final de ações obtidos na etapa de Preparação de Dados (Seção 4.1.2), de modo que 90% dos exemplos foram selecionados para treinamento e 10% para teste. Cada exemplo é composto por 7 entradas e 1 saída, conforme apresentado na Figura 17.

A entrada *sActionType* identifica o tipo de ação realizada (ex: clique, preenchimento de campo, etc). Já as entradas *sTag*, *sTagIndex* e *sXPath* servem para identificar o elemento

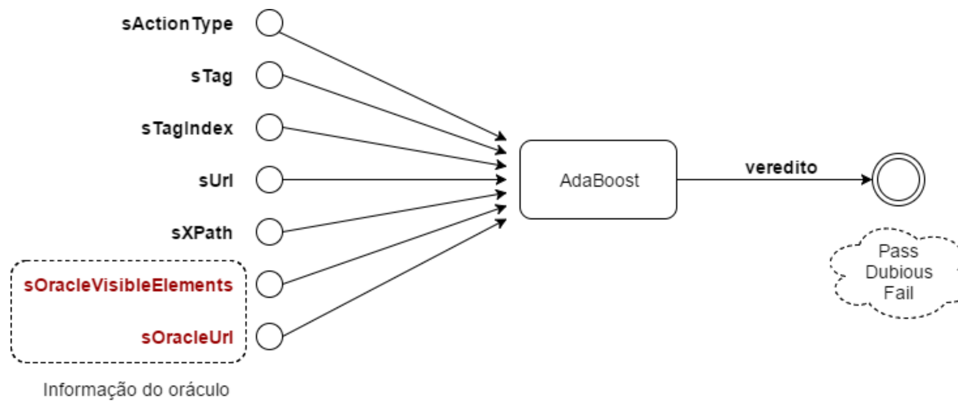


Figura 17 – Etapa de Aprendizagem de Máquina.

no qual a ação foi realizada (ex: botao, link, etc.). A entrada *sUrl* tem o objetivo de identificar a página da aplicação na qual a ação foi realizada. As entradas *sOracleVisibleElements* e *sOracleUrl* representam a informação do oráculo, ou seja, correspondem à informação que determina a corretude de execução de um determinado caso de teste. Finalmente, a saída *veredito* identifica a saída do caso de teste, se está correta ou existe falha na execução de cada caso de teste.

Após a realização do processo de treinamento, o classificador treinado é então testado utilizando o conjunto de testes, com o objetivo de analisar o desempenho do classificador na generalização do problema e com intuito de verificar se os resultados satisfazem os objetivos inicialmente propostos. Nesse trabalho o desempenho do classificador é verificado com base na análise das medidas apresentadas na Seção 2.3.

4.2.4 Ferramenta

De posse da abordagem proposta, foi desenvolvido um protótipo de uma ferramenta Web que contempla o método proposto neste trabalho. O protótipo está disponível para a comunidade científica de forma *open-source* em um repositório de código [1]. O protótipo foi desenvolvido com o intuito de facilitar a avaliação da abordagem proposta neste trabalho. Em trabalhos futuros esse protótipo será complementado para a criação da ferramenta de monitoramento do comportamento das aplicações.

A Figura 18 apresentam as principais tecnologias utilizadas no desenvolvimento da ferramenta, agrupadas nas camadas de *Front-end* e *Back-end*. O *Front-end* representa a interface visual com seus componentes que são manipulados pelo usuário, enquanto o *Back-end* responsável pelo processamento das requisições realizadas pelos clientes, além do armazenamento dos *logs* das aplicações em teste.

Para implementação do *Back-end* adotou-se a linguagem de programação JAVA

[1] Source-Code do protótipo: <https://github.com/Ronyerison/oracle-test.git>

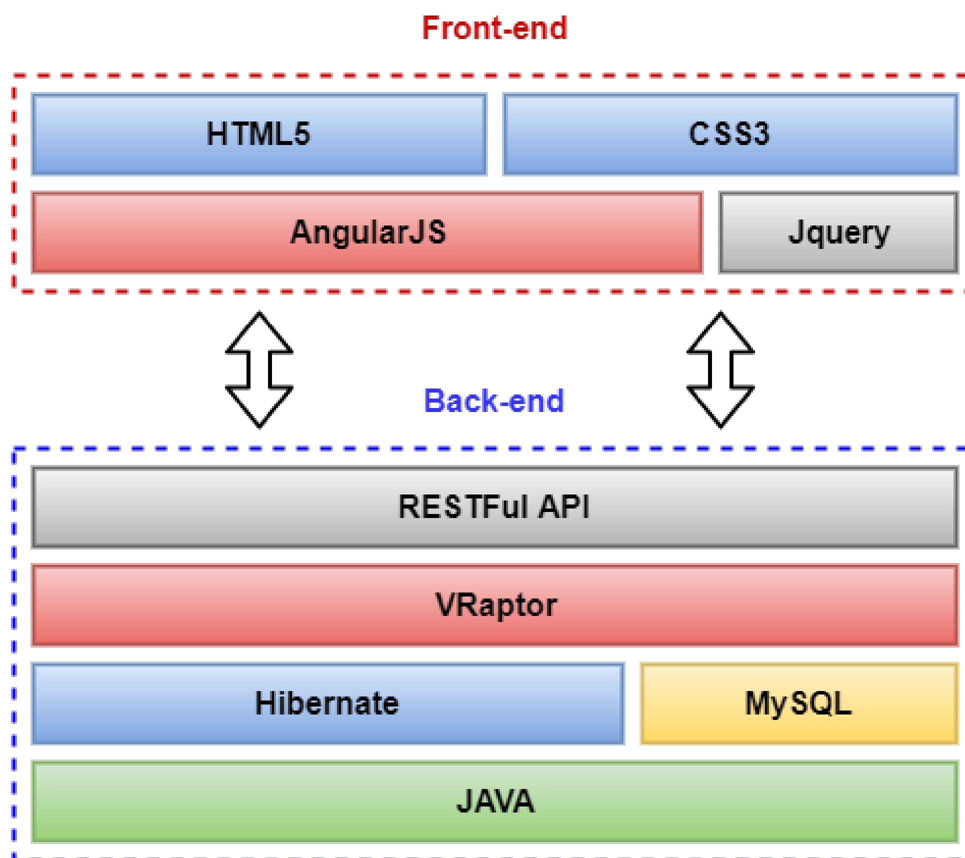


Figura 18 – Tecnologias utilizadas na implementação da ferramenta.

(GOSLING, 2000). O VRaptor[2] foi escolhido como *framework* para desenvolvimento da aplicação por possuir boa produtividade no desenvolvimento de aplicações Web, além ser adaptado ao padrão *RESTFul* e possuir boa documentação. O sistema de gerenciamento de banco de dados utilizado foi o MySQL[3] e o *framework* Hibernate[4] para facilitar o mapeamento objeto-relacional.

Os componentes visuais do *Front-end* foram desenvolvidos utilizando HTML5 e CSS3 (FRAIN, 2012). Além disso foi utilizado o *framework* AngularJS[5], pois é adaptado ao padrão *RESTFul* e possui boa documentação que facilita o desenvolvimento e aumenta a produtividade.

4.2.4.1 Funcionalidades da Ferramenta

Atualmente o protótipo da ferramenta desenvolvida apresenta um grupo de funcionalidades que foram desenvolvidas com o propósito de facilitar a avaliação da abordagem proposta neste trabalho. As funcionalidades desenvolvidas estão divididas em três grupos principais:

[2] VRaptor - <http://www.vraptor.org>
[3] MySQL - <http://www.mysql.com>
[4] Hibernate - <http://hibernate.org>
[5] AngularJS - <http://angularjs.org>

- **Gerenciamento de aplicações:** compreende um grupo de funcionalidades nos quais o usuário pode gerenciar as informações das aplicações sob teste. É possível realizar o cadastro de uma nova aplicação, visualizar as aplicações cadastradas, remover uma aplicação, etc.

- **Gerenciamento de ações:** compreende um grupo de funcionalidades para gerenciamento das ações capturadas das aplicações em testes. Consiste em um webservice que recebe requisições com dados em formato JSON contendo as ações capturadas enviadas pelas aplicações (etapa de captura de ações).

- **Gerenciamento de simulações:** compreende um grupo de funcionalidades para controle das simulações executadas em cada aplicação. Essas funcionalidades permitem o usuário gerar ações com falhas de forma aleatória (etapa de preparação de dados), realizar execuções dos algoritmos de aprendizagem de máquina (etapa de aprendizagem de máquina), visualizar gráficos de execuções (etapa de avaliação da aprendizagem)

A Figura 19 apresenta a página da ferramenta que lista as aplicações cadastradas por um usuário na ferramenta. A partir dessa tela é possível visualizar os dados de cada aplicação, editar informações, visualizar as simulações de cada aplicação, etc.

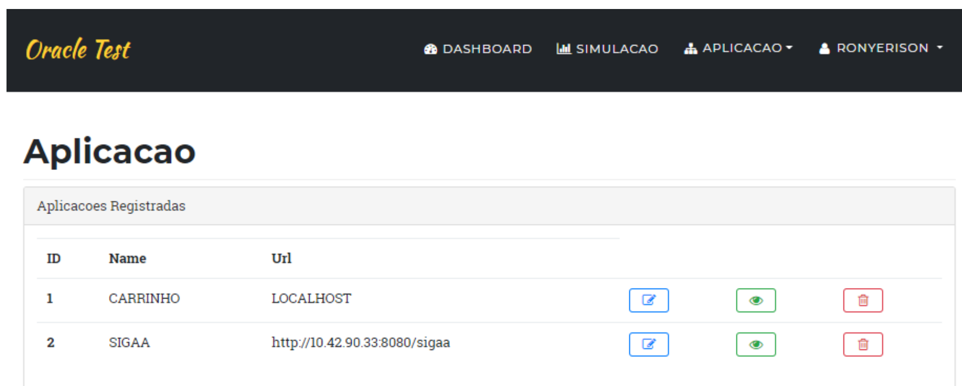
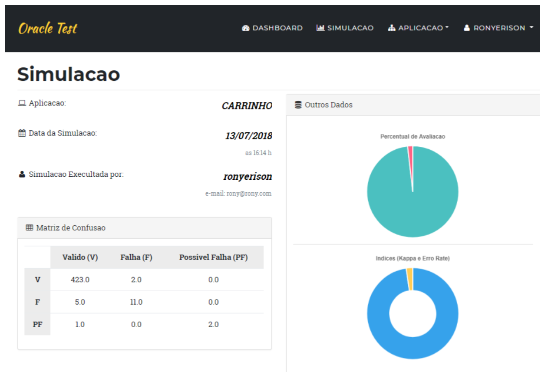


Figura 19 – Captura de tela da página de visualização de aplicações cadastradas.

Na Figura 20 é apresentada a página da ferramenta na qual o usuário pode visualizar a matriz de confusão e os gráficos de execução de uma simulação do oráculo em uma aplicação. Nessa página é possível analisar o desempenho da aprendizagem de máquina em cada simulação executada. Dessa forma, o usuário pode verificar os resultados de testes da abordagem com o grupo de ações capturadas e geradas pela abordagem.



(a) Captura de tela com resultados da matriz de confusão e medidas gerais.

(b) Captura de tela com resultados das medidas por classe.

Figura 20 – Captura de tela com resultados de uma simulação.

4.3 Considerações Finais

Este capítulo apresentou a abordagem proposta. Inicialmente, foi apresentada uma visão geral do método e posteriormente foram discutidas cada uma das etapas de forma individualizada, apresentando também as ferramentas utilizadas durante a implementação do método proposto. Após a apresentação do método, foram apresentadas as principais tecnologias utilizadas no decorrer do processo de desenvolvimento da ferramenta que encapsula o método proposto nesse trabalho, assim como, as funcionalidades desenvolvidas no protótipo da ferramenta. O próximo capítulo irá discutir as avaliações empíricas realizadas para mensurar o apoio do método na automação de oráculos de testes em aplicações Web.

Parte IV

Resultados e Discussões

5 Avaliação Experimental

Este capítulo apresenta os resultados das avaliações experimentais referente à abordagem proposta para automação do mecanismo de oráculo de teste. Para avaliar a abordagem proposta, quatro avaliações foram executadas. A primeira avaliação experimental foi executada com o objetivo de verificar a efetividade do método na identificação de falhas de software geradas aleatoriamente em um pequeno software (fictício). A segunda avaliação foi realizada com o propósito verificar a capacidade de localizar falhas inseridas por um especialista no mesmo software usado no primeiro experimento. Já a terceira avaliação foi realizada com inserção de falhas através de testes de mutação, com o objetivo de analisar o desempenho da abordagem proposta com falhas inseridas no SUT por meio da captura de ações realizadas em versões mutantes da aplicação.

Após a execução dos experimentos com software fictício, realizamos um novo experimento, usando como alvo uma aplicação real de grande porte (superior a 3 milhões de linhas de código), com o objetivo de validar os resultados obtidos nos primeiros experimentos, porém em um contexto bem maior.

Os dados dos experimentos realizados neste trabalho estão disponíveis em um diretório online[1], que contém arquivos com as ações usadas em cada experiência, bem como como um arquivo com uma versão mutante do projeto da aplicação Carrinho-de-Compras utilizada nos três experimentos preliminares.

5.1 Aplicações sob Teste

O software utilizado nas três primeiras avaliações da abordagem proposta foi uma aplicação Web fictícia, desenvolvida inicialmente durante a realização de um minicurso realizado na Universidade Federal do Piauí. A aplicação, denominada Carrinho-de-Compras, gerencia e mantém registros de compras de produtos, além de gerenciar e manter registros de clientes de um estabelecimento comercial.

A aplicação Carrinho-de-Compras foi desenvolvida em linguagem JAVA, na plataforma JAVA EE (*Java Platform, Enterprise Edition*), que é uma plataforma padrão para desenvolvimento de aplicações de grande porte e/ou para a internet. O desenvolvimento da aplicação foi realizado utilizando o *framework* JSF (*JavaServer Faces*), que é um *framework* Web baseado em Java que tem o propósito de simplificar o desenvolvimento de interfaces (páginas) de sistemas para Web, por meio de um modelo de componentes reutilizáveis.

1 Pacote de Dados dos Experimentos: <https://goo.gl/7B9Hhx>

O software utilizado na quarta avaliação experimental foi uma aplicação web que gerencia o controle de uma universidade. A aplicação mantém todos os registros de estudantes, professores, cursos, turmas, disciplinas, além diversas outras funcionalidades. O software é bastante conhecido e utilizado no Brasil e contém cerca de 3 milhões de linhas de código, considerando somente o *back-end* da aplicação (sem considerar páginas da aplicação e outros recursos).

5.2 Questões de Pesquisa

As avaliações experimentais realizadas neste trabalho foram propostas para responder a seguinte questão de pesquisa:

- **QP1**: A abordagem baseada em aprendizagem de máquina é eficaz para criação de um mecanismo de oráculo de teste?

Para avaliar a eficácia da abordagem como mecanismo de oráculo de teste foram usadas as medidas de Acurácia, Precisão, *Recall*, Especificidade, *F-Measure* e AUC. Medidas comumente utilizadas na literatura para avaliar o desempenho de técnicas baseadas em aprendizagem de máquina na solução de diversos problemas (SOKOLOVA; LAPALME, 2009).

5.3 Operação da Primeira Avaliação

Inicialmente, o componente de captura foi inserido em todas as interfaces (páginas) da aplicação Carrinho-de-Compras para que as ações realizadas por usuários na aplicação fossem capturadas e enviadas para o servidor responsável pelo armazenamento. Esta atividade corresponde à etapa de Captura de Ações (Seção 4.1.1) do método proposto nesse trabalho. Nessa etapa foi realizada a captura de um total de 637 ações realizadas por usuários da aplicação.

Após a captura de ações, assumiu-se que as 637 ações capturadas correspondem a casos de testes que expressam o comportamento correto da aplicação. O conjunto de ações foi utilizado como base para o aprendizado supervisionado, pois serviu de exemplo sobre como a aplicação deve funcionar. Este conjunto de ações representou o conjunto de casos de testes pertencentes à classe “Válido” do algoritmo de aprendizagem de máquina.

Na etapa de Preparação de Dados, uma análise do ganho de informação foi realizada para identificar os atributos (informação) que não eram relevantes para serem usadas na aprendizagem de máquina. Baseado nesta análise, os atributos “tempo”, “aplicação” e “quem fez a ação” foram removidos. Como esperado, o ganho de informação para esses

atributos foi nulo. A Tabela 10 apresenta o resultado do ganho de informação dos atributos capturados na etapa de Captura de Ações.

Tabela 10 – Ganho de Informação dos atributos capturados.

Atributo	**Ganho de Informação**
sOracleVisibleElements	0.14373
sOracleUrl	0.03248
sXPath	0.02725
sTagIndex	0.02184
sUrl	0.01930
sTag	0.01405
sActiontype	0.00647

É importante destacar que a análise do ganho de informação deve ser realizada em relação a cada aplicação individualmente. Por exemplo, nas avaliações experimentais realizadas neste trabalho, a informação da pessoa que executou a ação obteve valor nulo, uma vez que as funcionalidades analisadas nas aplicações em teste não exigiram um controle de acesso diferente por usuário. No entanto, para outras aplicações, essas informações podem ser relevantes para a identificação do comportamento do software.

Após a análise do ganho de informação, foi realizada a inserção na base de dados das ações com falhas que iriam compor o potencial comportamento incorreto do software. Nesta etapa, foi gerada uma cópia de aproximadamente 30% (184 instâncias) do total de ações capturadas.

A partir de um caso de teste válido, que representa um comportamento correto do sistema, foram geradas cópias e realizadas modificações associadas aos atributos que compõem a informação de oráculo: *sOracleVisibleElements* e *sOracleUrl*. Por exemplo, um caso de teste que contém os valores “240” e “https://aplicacao.com/login”, para os atributos *sOracleVisibleElements* e *sOracleUrl*, respectivamente, foi modificado para conter os valores “30” e “https://aplicacao.com/compra”. Essas modificações geraram um caso de teste com falha.

É importante destacar que as modificações citadas acima foram realizadas de forma aleatória, no entanto, dentro dos valores possíveis da aplicação. Isto é, não foram gerados números negativos para o atributo *sOracleVisibleElements*, assim como, os valores gerados para o atributo *sOracleUrl* foram URL’s da aplicação, mas diferentes da URL correta para o caso de teste. Dos 184 casos de testes, em 131 foram realizadas modificações contendo esse tipo de falha. Esse grupo de ações corresponde aos casos de testes pertencentes à classe “Falha” do algoritmo de aprendizagem de máquina.

O restante dos casos de testes (53 instâncias) foram modificados de forma a conter dados que os considere como pertencentes à classe “Possível Falha”. A abordagem aqui proposta identifica tais casos de teste e apenas sugere que são suspeitos de conterem falhas.

As modificações para esses casos de testes foram realizadas apenas no atributo *sOracleVisibleElements* em um intervalo próximo ao valor correto. Seguindo o mesmo exemplo do item anterior, um caso de teste que contém os valores "240" e "https://aplicacao.com/login" para os atributos *sOracleVisibleElements* e *sOracleUrl*, respectivamente, modificado para "260" no atributo *sOracleVisibleElements* e permanecendo igual em relação ao valor do atributo *sOracleUrl* foi considerado um caso de teste com possível falha.

Na etapa de aprendizagem de máquina (Seção 4.1.3) foi utilizada a ferramenta WEKA (HALL et al., 2009), em conjunto com o plugin AUTO-WEKA (THORNTON, 2014). Inicialmente, foi executado o AUTO-WEKA para identificar o melhor algoritmo e seus hiper-parâmetros para solução do problema, levando em conta o valor para taxa de erro. Como resultado dessa execução o plugin identificou o *AdaBoostM1* em conjunto com o JRIP[2] com menor taxa de erro na solução do problema.

Após a execução do AUTO-WEKA, o algoritmo selecionado foi executado na WEKA para verificação dos resultados da automação do oráculo de testes para a aplicação Carrinho-de-Compras. A Tabela 11 apresenta os parâmetros de configuração dos algoritmos *AdaBoostM1* e *JRIP* utilizados no experimento realizado.

Tabela 11 – Hiper-parâmetros de configuração dos algoritmos de AM.

Algoritmo	**Nome do Parâmetro**	**Valor**
AdaBoostM1	batchSize	100
	debug	false
	doNotCheckCapabilities	false
	numDecimalPlaces	2
	numIterations	10
	seed	1
	useResampling	false
	weightThreshold	100
	classifier	JRIP
JRIP	batchSize	100
	checkErrorRate	true
	debug	false
	doNotCheckCapabilities	false
	folds	3
	minNo	2.0
	numDecimalPlaces	2
	optimizations	2
	seed	1
	usePruning	true

O *Cross-Validation* foi um método originalmente empregado para avaliar a validade preditiva das equações de regressão linear (MOSIER, 1951). Essa técnica é importante para verificar como o algoritmo generaliza o conhecimento independente do conjunto de dados

[2] Implementação do algoritmo IREP na ferramenta Weka

analisado (BROWNE, 2000). Em um problema de predição, um modelo geralmente recebe um conjunto de dados conhecidos no qual o treinamento é executado e um conjunto de dados desconhecidos ao qual o modelo é testado. Na validação cruzada, o conjunto de dados é dividido aleatoriamente em k subconjuntos mutuamente exclusivos. O modelo é treinado e testado k vezes, cada vez que é treinado com um subconjunto e testado com outro. Ao final do processo, a acurácia é calculada em relação ao total de exemplos classificados corretamente pelo modelo (KOHAVI et al., 1995). Foi decidido que o *Cross-Validation* deveria ser adotado durante a execução do algoritmo *AdaBoostM1* para automação do oráculo de teste. No primeiro experimento realizado com o Carrinho-de-Compras foi utilizado o *k-fold cross-validation* com $k = 10$.

5.4 Operação da Segunda Avaliação Experimental

Após a primeira avaliação, decidiu-se realizar uma segunda avaliação experimental utilizando o software Carrinho-de-Compras com o objetivo de validar a abordagem proposta em relação à geração de falhas aleatórias inseridas no conjunto de casos de teste. Neste segundo experimento, foi realizada a inserção de falhas por um desenvolvedor da aplicação, tendo sido capturadas as ações para testar o classificador gerado no primeiro experimento.

O objetivo foi identificar se as falhas geradas aleatoriamente na primeira avaliação representavam falhas reais cometidas pelos desenvolvedores durante o desenvolvimento do software. Dessa forma, foi utilizado o classificador já treinado no primeiro experimento e foi realizada apenas a etapa de teste do algoritmo, com o conjunto de ações capturadas com as falhas inseridas pelo desenvolvedor.

Um total de dezoito (18) falhas diferentes foram inseridas na aplicação. O desenvolvedor inseriu as falhas, com base em sua experiência como desenvolvedor. Após a inserção, as ações nos elementos contendo as falhas foram capturadas e um conjunto de testes foi gerado para o classificador obtido no primeiro experimento.

5.5 Operação da Terceira Avaliação Experimental

A terceira avaliação da abordagem foi realizada com o Carrinho-de-Compras utilizando falhas inseridas por meio de testes de mutação. A geração das classes mutantes da aplicação foi realizada com o suporte da ferramenta Pitest[3] e Pitest Export Plugin[4].

Inicialmente, foram selecionadas 3 classes JAVA do Carrinho-de-Compras que continham as regras de negócios da aplicação. Depois de escolher as classes, gerou-se os mutantes usando a ferramenta Pitest. Os operadores de mutação utilizados no experimento

[3] Pitest Website: <http://pitest.org/>
[4] Export Plugin Repository: <https://github.com/pitest/export-plugin>

foram o *Void Method Call Mutator*, responsável por remover a chamada de métodos e *Negate Conditional Mutator*, que realiza mudanças nos operadores condicionais da classe. Esses operadores foram selecionados porque o uso conjunto dos dois não interferiam nas classes mutantes produzidas entre eles. O uso de outros operadores juntos poderia substituir os erros produzidos entre eles. No total, 9 classes mutantes contendo diferentes erros foram geradas. Com base nas classes geradas pela ferramenta Pitest, uma nova versão do Carrinho-de-Compras contendo todos os erros foi criada.

O experimento foi iniciado pela captura das ações válidas realizadas com a versão funcional do carrinho de compras, tendo sido capturadas 425 ações válidas. Após a captura inicial, uma nova captura de ações foi realizada com a versão mutante do aplicativo, sendo que neste segundo estágio foram capturadas 16 ações com falhas e 3 ações com possíveis falhas.

No final da captura de ações válidas e com falha, foi realizado um novo treinamento e teste de algoritmos de aprendizado de máquina. Utilizou-se a mesma estratégia do primeiro experimento nesta etapa da abordagem. O *k-fold cross-validation* foi usado com $k = 10$. O conjunto de ações foi aleatoriamente ordenado e dividido em 10 partes.

5.6 Operação da Quarta Avaliação Experimental

Finalmente, com a realização dos experimentos foi possível obter resultados considerados satisfatórios em relação a aplicação da abordagem para automação do oráculo. No entanto, como os experimentos foram realizados em testes com uma aplicação de pequeno porte, decidiu-se realizar uma nova avaliação experimental em uma aplicação real de grande porte.

Nesta quarta avaliação, foi utilizada a mesma estratégia do primeiro. Este experimento foi realizado com o objetivo de verificar se os resultados obtidos com uma aplicação de pequeno porte também poderiam ser constatados em um cenário real em testes com uma aplicação de grande porte.

Um total de 22.000 ações foram usadas neste experimento, sendo 20.000 para treinamento e 2.000 para testes. O conjunto de treinamento consistiu de 14.000 ações válidas obtidas pela captura de ações realizadas por usuários da aplicação, além de 4.000 ações com falhas e outras 2.000 com possíveis falhas. O subconjunto teste foi formado aleatoriamente a partir da seleção de um conjunto de ações diferentes das utilizadas no treinamento. O total de ações de teste pertencentes a cada classe é apresentada na matriz de confusão com os resultados deste experimento (Tabela 17).

5.7 Resultados das Avaliações Experimentais

As Tabelas 12 e 13 apresentam os resultados coletados durante a execução da primeira avaliação experimental realizada com o "Carrinho-de-Compras". Na Tabela 12 é apresentada a matriz de confusão que contém os números de casos de testes pertencentes a cada uma das classes e o número de casos de testes classificados em cada uma delas pelo algoritmo de *AdaBoostM1*.

Tabela 12 – Matriz de confusão da primeira avaliação.

		Esperado		
		Válida	Falha	Possível Falha
Predito	Válida	633	2	2
	Falha	15	109	7
	Possível Falha	22	4	27

Com base na análise dos resultados do experimento, é possível perceber que o *AdaBoostM1* obteve um bom desempenho para automação do oráculo de testes. A diagonal principal da matriz de confusão representa a quantidade de casos de testes classificados corretamente, sendo 633 casos de testes válidos, 109 casos de testes com falhas e em outros 27 com possíveis falhas (duvidosos). Após a construção da matriz de confusão foram realizados os cálculos das medidas que foram utilizadas para avaliação do desempenho do algoritmo (Seção 2.3).

Os cálculos das medidas de **Acurácia – A** e **Erro – E** foram realizados em relação ao total de instâncias classificadas, obtendo como resultado os seguintes valores: $A = 0.93$ e $E = 0.07$. Esses resultados indicam que o algoritmo obteve um resultado muito bom para a automação do oráculo de testes, considerando a base de dados utilizada no experimento. No entanto, é necessário identificar se o algoritmo classifica corretamente os exemplos pertencentes a cada uma das classes do problema, ou seja, verificar se o algoritmo identifica corretamente os casos de testes válidos, com falhas e possíveis falhas. Os resultados das medidas calculadas por classe são apresentados na Tabela 13.

Tabela 13 – Resultado do calculo das medidas da primeira avaliação experimental.

Classe	Taxa de VP	Taxa de FP	P	*R*	*FM*	Curva ROC
Válido	0.994	0.201	0.945	0.994	0.969	0.950
Falha	0.832	0.009	0.948	0.832	0.886	0.990
Possível Falha	0.509	0.012	0.750	0.509	0.607	0.859
Média Ponderada	**0.937**	**0.158**	**0.933**	**0.937**	**0.932**	**0.951**

Com base nos dados apresentados na Tabela 13, pode-se perceber que o algoritmo obteve resultados satisfatórios para classificação dos exemplos pertencentes a cada uma

das classes. Para os casos de testes pertencentes às classes Válido e Falha, o algoritmo obteve um melhor desempenho. Já em relação a classe Possível Falha, o resultado das medidas calculadas foi menor. Esse fato pode ser explicado com base na análise da matriz de confusão (Tabela 12). Nela é possível perceber que existem 22 exemplos pertencentes a classe Possível Falha que foram classificados como Válidos, isso se dá porque os valores que compõem a informação do oráculo (*sOracleVisibleElements* e *sOracleUrl*) são próximos dos valores considerados corretos (exemplo apresentado na Seção 4.1.2).

Na segunda avaliação experimental realizada com o "Carrinho-de-Compras" utilizou-se o classificador treinado com os dados do primeiro experimento. Foi realizada apenas a etapa de teste do algoritmo com as falhas inseridas pelo desenvolvedor. Nesse experimento, foi inserido um total de 18 falhas no software e como resultado o algoritmo identificou corretamente 9 falhas. Outras 5 falhas o algoritmo classificou como ações válidas e 4 foram identificadas como possíveis falhas. A Tabela 14 apresenta o resultado dos cálculos das medidas para o segundo experimento.

Tabela 14 – Resultados das medidas da segunda avaliação experimental.

Classe	**Taxa de VP**	**P**	***R***	***FM***
Falha	0,720	1,000	0,720	0,8387

É importante destacar que o valor para a medida de especificidade obtido foi 0.0, pois o experimento só foi realizado com casos de testes pertencentes à classe "Falha". Por essa mesma razão não foi calculada a área sob a curva ROC. Como pode ser percebido na tabela, o algoritmo classificou corretamente 72% das falhas inseridas, valores que são representados nas medidas das taxas de Verdadeiros Positivos (VP) e *Recall (R)*.

Com base na análise dos resultados das duas avaliações é possível perceber que: apesar da taxa de acerto não ser tão alta na segunda avaliação, a abordagem foi bastante eficaz na identificação das falhas, uma vez que, na maioria dos casos que foram indicadas falhas pelo algoritmo realmente os casos de testes correspondiam a falhas na aplicação. Isso pode ser percebido na baixa taxa de Falsos Positivos das classes "Falha" e "Possível Falha" no primeiro experimento.

A Tabela 15 apresenta a matriz de confusão com os resultados da terceira avaliação experimental realizada com testes de mutação. Analisando a matriz de confusão é possível perceber que o algoritmo classificou corretamente 423 ações válidas, 11 ações com falhas e 2 ações com possíveis falhas. De modo geral o algoritmo obteve um desempenho de aproximadamente 98 por cento (98%) de acerto na classificação das ações capturadas. O algoritmo errou ao realizar a classificação de 2 ações válidas, 5 falhas e 1 possível falha, totalizando um percentual de aproximadamente 2 por cento (2%) de erros.

A Tabela 16 apresenta os resultados da medidas para cada classe na terceira avaliação. Com base na análise das medidas é possível perceber que o algoritmo foi muito

Tabela 15 – Matriz de Confusão da terceira avaliação experimental

		Esperado		
		Válida	Falha	Possível Falha
Predito	Válida	423	2	0
	Falha	5	11	0
	Possível Falha	1	0	2

preciso na identificação das ações pertencentes a cada classe (Altas taxas de precisão e baixas taxas de falsos positivos). Dessa forma, considerando o contexto do terceira avaliação pode-se destacar que a abordagem proposta obteve resultados satisfatórios na solução do problema.

Tabela 16 – Resultado do cálculo das medidas da terceira avaliação experimental.

Classe	**Taxa de VP**	**Taxa de FP**	**P**	***R***	***FM***	**Curva ROC**
Válida	0.995	0.315	0.986	0.995	0.990	0.997
Falha	0.687	0.004	0.846	0.687	0.758	0.630
Possível Falha	0.666	0.0	1.0	0.666	0.8	0.673
Média	0.940	0.141	0.937	0.940	0.937	0.949

Com o objetivo de aprofundar a análise da abordagem foi conduzido uma quarta avaliação experimental da abordagem com uma aplicação real de controle de estudantes. No experimento foi utilizado um total de 20000 ações para treinamento do algoritmo e 2000 ações para teste. A Tabela 17 mostra a matriz de confusão com os resultados do teste com o algoritmo.

Tabela 17 – Matriz de Confusão com os resultados do quarto experimento.

		Esperado		
		Válida	Falha	Possível Falha
Predito	Válida	1464	7	3
	Falha	215	84	0
	Possível Falha	180	8	39

Com base na análise da matriz de confusão é possível observar que 1464 ações foram corretamente classificadas para a classe "Válida", 84 para a classe "Falha" e 39 para a classe "Possível Falha". É possível observar também que a maioria dos erros foram relacionadas a identificação das classes "Falha" e "Possível Falha", nas quais o algoritmo classificou como "Válida", ocorrendo em um total de 395 casos. Este fato pode ser explicado pelo desbalanceamento em relação ao número de ações treinadas para cada uma das classes. Contudo, como neste experimento pretendíamos aplicar a abordagem em um cenário mais próximo da realidade, optou-se por não usar nenhuma técnica de balanceamento de classes,

pois o número de funcionalidades desenvolvidas corretamente não é equivalente ao número de falhas considerando um contexto real de desenvolvimento de software.

O cálculo das medidas de Acurácia (A) e Erro (E) foi realizado em relação ao total de exemplos classificados, resultando nos seguintes valores: $A = 0.79$ (79%) and $E = 0.21$ (21%). A Tabela 18 apresenta o resultados os cálculos das medidas para cada uma das classes. Foi possível observar que o algoritmo obteve altas taxas de precisão para cada uma das classes, indicando uma alta probabilidade da classificação do algoritmo estar correta. A medida de *Recall* para as classes "Falha" e "Possível Falha" ficou abaixo do esperado, esse fato deve-se ao alto numero de erros em exemplos pertencentes à essas classes que foram classificados pelo algoritmo como "Válidos". Acredita-se que com a utilização de técnicas de pré-processamento de dados pode ajudar a melhorar a identificação dos exemplos quando o conjunto de dados for desbalanceado, como foi o caso deste quarto experimento. Pretende-se realizar novas avaliações com diferentes aplicações para investigação das melhorias propostas.

Tabela 18 – Resultado do cálculo das medidas da quarta avaliação experimental.

Classe	**Taxa de VP**	**Taxa de FP**	**P**	***R***	***FM***	**Curva ROC**
Válida	0.993	0.762	0.787	0.993	0.878	0.849
Falha	0.281	0.009	0.848	0.281	0.422	0.537
Possível Falha	0.172	0.001	0.929	0.172	0.290	0.459
Média	0.793	0.563	0.812	0.793	0.743	0.758

De acordo com as medidas calculadas (Etapa de avaliação da aprendizagem), a eficácia da abordagem foi considerada satisfatória para resolver o problema e apresentou indicações de que a abordagem pode ser usada para a automação do mecanismo de oráculo de testes. É possível notar que a abordagem sempre obteve altas taxas de precisão em relação às classes "Falha" e "Possível Falha", uma vez que o foco principal do oráculo de teste é a identificação de problemas de software. Além disso, a abordagem obteve baixas taxas de falsos negativos para essas classes, portanto, há uma alta probabilidade de que a abordagem indique uma falha que é realmente um problema em um software.

Por fim, com a realização dos experimentos, foi possível perceber que a abordagem proposta funciona bem mesmo com conjuntos de dados desbalanceados (o que ocorre no cenário real de desenvolvimento de software), como pode ser observado nos resultados obtidos para a medida F-Measure calculada em cada experimento realizado neste trabalho.

5.8 Ameaças à Validade

Uma questão fundamental a respeito dos resultados de um estudo experimental é o quão válido são os resultados obtidos. Dessa forma, é importante analisar os riscos

relacionados aos experimentos e propor alternativas para tentar contornar os problemas que possam ameaçar à validade dos resultados obtidos (WOHLIN et al., 2012).

5.8.1 Validade Externa

A validade externa está relacionada ao risco de generalização dos resultados, pois no caso de estudos experimentais alguns fatores podem ameaçar a validade do estudo e não permitir a generalização dos resultados obtidos nos experimentos.

Uma das limitações deste trabalho está relacionada ao software utilizado nos experimentos preliminares. Conforme discutido na Seção 5.1, os três primeiros experimentos foram realizados com uma aplicação fictícia, que contém um conjunto limitado de funcionalidades. Dessa forma, o conjunto de ações capturadas para os experimentos foi restrito a um pequeno grupo de funcionalidades. Contudo, para tentar contornar esse problema, utilizamos o método de validação cruzada durante a etapa de aprendizagem de máquina para assegurar os resultados obtidos independente do conjunto de dados avaliado pelo classificador. Além disso, realizamos um quarto experimento com uma aplicação real de grande porte conhecida e utilizada em universidades no Brasil.

Em trabalhos futuros pretende-se realizar novos experimentos com aplicações reais e assegurar os resultados obtidos nesta pesquisa.

5.8.2 Validade Interna

Este aspecto está relacionado à validade dos resultados identificando a relação causa-efeito (WOHLIN et al., 2012), ou seja, uma ameaça relacionada ao controle do processo de experimentação para coletar os dados analisados. Outra limitação deste trabalho refere-se às ações aleatórias geradas pelo método proposto. Para atenuar essa ameaça, um segundo experimento foi conduzido em que as falhas foram inseridas diretamente na aplicação por um especialista e capturadas para avaliação do classificador, e um terceiro experimento foi realizado, usando o teste de mutação para criar novas versões do software a serem usadas como um SUT.

Outro fator que pode ameaçar a qualidade dos resultados obtidos está relacionado à configuração de hiper parâmetros do algoritmo ML utilizado. Para contornar essa ameaça, foi utilizado o plugin AUTO-WEKA que realiza diferentes execuções dos algoritmos de aprendizagem com diversas configurações.

5.8.2.1 Validade de Conclusão

A validade da conclusão está relacionada à capacidade de chegar a conclusões de acordo com os resultados obtidos a partir da aplicação do tratamento. Apesar do

conjunto limitado de dados capturados nos primeiros experimentos, os resultados mostraram indicações da adequação da abordagem na solução do problema.

Para tentar contornar essa ameaça foi realizado um quarto experimento com uma aplicação real de grande porte utilizando a geração de falhas de forma aleatória. Com a execução de um estudo de caso com aplicações mais robustas e aplicação da abordagem em um cenário real de desenvolvimento de software espera-se aumentar a confiabilidade dos resultados obtidos.

6 Conclusões e Trabalhos Futuros

A automação em oráculos de testes vem se mostrando cada vez mais importante na redução de esforço e custo na realização de Testes de Software. Uma vez que essa etapa tem sido considerada uma das mais onerosas no processo de desenvolvimento de software, consumindo em até 50% dos recursos de desenvolvimento (MYERS; SANDLER; BADGETT, 2011). Conforme, análise inicial da área percebeu-se que o número de pesquisas relacionadas a automação em oráculos de testes vêm crescendo bastante nos últimos anos. Entretanto, com a realização de um Mapeamento Sistemático foi possível realizar uma análise mais profunda dessa linha de pesquisa e percebeu-se lacunas passíveis de exploração, por exemplo, falta de trabalhos que combinassem a utilização de logs de uso de aplicações com aprendizagem de máquina.

Considerando esse contexto, o objetivo geral deste trabalho é propor uma abordagem para automação do mecanismo de oráculo de testes. Essa abordagem foi desenvolvida com o objetivo de reduzir custos e erros na realização da atividade de Teste de Software, pois o oráculo de testes ainda é considerado um gargalo na realização de tal atividade, uma vez que, quando realizada de forma manual é necessário um desenvolvedor para verificação do comportamento dos softwares em teste.

Para alcançar o objetivo geral deste trabalho, decidiu-se adotar uma metodologia composta pelas seguintes ações: elaborar um estudo detalhado da área, evidenciando as principais técnicas utilizadas na literatura para automação de um oráculo de teste; realizar um mapeamento sistemático, com o objetivo de identificar os principais trabalhos dessa linha de pesquisa; desenvolver uma nova abordagem para automação do mecanismo de oráculo de testes; implementar um protótipo de uma ferramenta que encapsule a abordagem proposta; avaliar a abordagem proposta por meio de estudos empíricos e sumarizar os resultados obtidos para publicação em eventos científicos.

A abordagem para automação do mecanismo de oráculo de testes (principal resultado produzido por esta pesquisa) foi elaborada, implementada (encapsulada em um protótipo de uma ferramenta) e avaliada em cenário fictício e real obteve resultados satisfatórios na solução do problema abordado. Essa abordagem se mostrou eficaz como mecanismo de oráculo de testes e portanto poderá ser utilizada para apoiar a realização de testes de regressão de forma contínua em aplicações executando em ambientes de produção. Dessa forma, considerando a metodologia proposta para alcançar o objetivo geral da pesquisa, destaca-se como principais contribuições do trabalho:

- **Realização de um mapeamento sistemático:** desenvolveu-se um estudo secundário com o objetivo de construir conhecimento a partir das publicações relacionadas

à pesquisas sobre a automação em oráculos de testes (Capítulo 3). A descrição da metodologia possibilita a replicação desse tipo de estudo e os resultados facilitam a escolha de um método para auxiliar a realização de testes de software em determinados tipos de softwares e serviram de base para o direcionamento de esforços durante a construção da abordagem;

- **Criação de uma abordagem para automação de oráculos de testes:** foi desenvolvida uma nova abordagem baseada em aprendizagem de máquina para automação de oráculos de testes (Capítulo 4). A abordagem foi desenvolvida com o objetivo de apoiar a realização de testes de regressão em aplicações voltadas para a plataforma web.

- **Desenvolvimento de um protótipo de uma ferramenta encapsulando a abordagem:** foi desenvolvido um protótipo de uma ferramenta web que encapsula a abordagem proposta. Esse protótipo será a base para um novo projeto que visa a construção de uma ferramenta para monitoramento de testes de regressão de forma contínua em softwares em ambiente de produção e obter *feedback* sobre os riscos de negócios associados a um software no ambiente de produção o mais rápido possível.

- **Avaliação da abordagem:** realizou-se quatro estudos empíricos para avaliar a abordagem proposta (Capítulo 5). O primeiro experimento foi realizado utilizando falhas inseridas de forma aleatória no conjunto de ações. O segundo experimento teve como objetivo principal identificar se as falhas aleatórias geradas no primeiro representavam um cenário real de falhas cometidas por desenvolvedores. O terceiro experimento foi realizado em conjunto com testes de mutação, no qual foi gerada uma versão mutante de uma aplicação sob teste para captura das ações com falhas. Finalmente, o quarto experimento foi realizado com uma aplicação de grande porte com o objetivo de comprovar os resultados obtidos nos primeiros experimentos.

Como consequência da realização deste trabalho, obteve-se uma publicação que contempla o núcleo dessa pesquisa. Foi aprovado para publicação no XXXII Simpósio Brasileiro de Engenharia de Software um artigo contendo a abordagem proposta para automação do mecanismo de oráculo de teste, conforme apresentado a seguir:

- Braga R.; Neto, P. S.; Rabêlo, R.; Santiago, J. R. R.; Souza, M. **A Machine Learning Approach to Generate Test Oracles**. XXXII Simpósio Brasileiro de Engenharia de Software (SBES). 2018. (aceito para publicação)

6.1 Desafios e Limitações

Apesar dos bons resultados obtidos até o momento, este trabalho ainda apresenta alguns desafios e limitações. O maior deles refere-se a avaliação do método proposto. Foram realizadas diferentes avaliações com uma aplicação de pequeno porte e foi realizada uma avaliação da abordagem utilizando falhas geradas de forma aleatória em uma aplicação de grande porte. Não foi possível realizar avaliações utilizando diferentes estratégias de geração de falhas com uma aplicação de grande porte. Dessa forma, pretende-se realizar em trabalhos futuros novas avaliações em diferentes aplicações em cenário real.

Uma limitação da abordagem refere-se ao fato de que ela não está adaptada a determinados algoritmos de pré-processamento de dados, por exemplo, balanceamento de dados. Essa questão merece uma investigação profunda para tentar aprimorar os resultados obtidos na solução do problema.

Outra limitação está relacionada à implementação da ferramenta para monitoramento de testes de regressão de forma contínua. Atualmente, a ferramenta encapsula a abordagem proposta, incluindo o armazenamento de ações capturadas de aplicações em testes, geração de falhas aleatórias, além das técnicas de aprendizagem de máquina e relatórios de execuções do oráculo de teste. O protótipo da ferramenta está disponível em repositório de código aberto. No entanto, para que seja possível disponibilizar a ferramenta hospedada em um servidor ainda é necessário o desenvolvimento de outras funcionalidades, além de uma avaliação robusta da ferramenta para assegurar a integridade e segurança das informações dos usuários.

Por fim, pode-se notar que mesmo com todo trabalho realizado até o momento, ainda há espaço para melhorias e trabalhos futuros. Dessa forma, tais limitações representam novas oportunidades de pesquisa.

6.2 Trabalhos Futuros

Considerando as limitações apresentadas na Seção 6.1, foram definidas algumas idéias para trabalhos futuros:

- **Novas avaliações da abordagem:** este trabalho apresenta quatro estudos empíricos. Os resultados indicam uma adequação quanto a aplicação da abordagem para solução do problema. Entretanto, essa conclusão não pode ser generalizada para o ambiente industrial sem mais avaliações. Dessa forma, ressalta-se a importância da realização de novas avaliações com diferentes aplicações em cenário real. Além disso, destaca-se a realização de novas avaliações empíricas da abordagem com o objetivo de realizar uma análise comparativa com a utilização de diferentes técnicas de aprendizagem de máquina;

- **Desenvolvimento da ferramenta:** foi desenvolvido neste trabalho um protótipo de uma ferramenta encapsulando a abordagem proposta, contudo, a ferramenta ainda apresenta um conjunto limitado de funcionalidades. Dessa forma, em novos trabalhos é possível incluir novas funcionalidades na ferramenta para torná-la robusta para apoio à realização de testes de regressão em aplicações web. Por exemplo, desenvolvimento de funcionalidades para notificação dos desenvolvedores à respeito de possíveis falhas nas aplicações sob teste, construção de relatórios de monitoramento das aplicações, dentre outras.

- **Revisão sistemática:** neste trabalho foi realizado um mapeamento sistemático referente aos trabalhos que visam automatizar o mecanismo de oráculo de teste. No entanto, esse estudo tem caráter quantitativo e não qualitativo. Dessa forma, é interessante realização de uma revisão sistemática estendendo o mapeamento para avaliar os trabalhos de forma qualitativa. Além disso, destaca-se a realização de uma extensão da pesquisa para englobar uma atualização dos trabalhos em relação ao período.

Referências

AGARWAL, D. et al. A comparative study of artificial neural networks and info-fuzzy networks as automated oracles in software testing. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, IEEE, v. 42, n. 5, p. 1183–1193, 2012. Citado na página 99.

ALMAGHAIRBE, R.; ROPER, M. Building test oracles by clustering failures. In: IEEE PRESS. *Proceedings of the 10th International Workshop on Automation of Software Test*. [S.l.], 2015. p. 3–7. Citado 2 vezes nas páginas 42 e 98.

ALMAGHAIRBE, R.; ROPER, M. Separating passing and failing test executions by clustering anomalies. *Software Quality Journal*, Springer, p. 1–38, 2016. Citado 3 vezes nas páginas 33, 42 e 97.

ANTONIE, M.-L.; ZAIANE, O. R.; COMAN, A. Application of data mining techniques for medical image classification. In: SPRINGER-VERLAG. *Proceedings of the Second International Conference on Multimedia Data Mining*. [S.l.], 2001. p. 94–101. Citado na página 3.

ARANTES, A. O.; SANTIAGO, V. A. de; VIJAYKUMAR, N. L. On proposing a test oracle generator based on static and dynamic source code analysis. In: IEEE. *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security-Companion, 2015. (QRS-C 2015)*. [S.l.], 2015. p. 144–152. Citado 2 vezes nas páginas 42 e 97.

ARISS, O. E. et al. A systematic capture and replay strategy for testing complex gui based java applications. In: IEEE. *Proceedings of the Seventh International Conference on Information Technology: New Generations, 2010. (ITNG 2010)*. [S.l.], 2010. p. 1038–1043. Citado na página 100.

BAHAROM, S.; SHUKUR, Z. Utilizing an abstraction relation document in grey-box testing approach. In: IEEE. *Proceedings of the International Conference on Electrical Engineering and Informatics, 2009. (ICEEI 2009)*. [S.l.], 2009. v. 1, p. 304–308. Citado na página 100.

BAI, X. et al. Semantic-based test oracles. In: IEEE. *Proceedings of the IEEE 35th Annual Computer Software and Applications Conference, 2011. (COMPSAC 2011)*. [S.l.], 2011. p. 640–649. Citado na página 100.

BARR, E. T. et al. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, IEEE, v. 41, n. 5, p. 507–525, 2015. Citado 5 vezes nas páginas 5, 13, 22, 28 e 34.

BAUERSFELD, S. et al. Evaluating the testar tool in an industrial case study. In: ACM. *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. [S.l.], 2014. p. 4. Citado na página 98.

BERGSTRA, J. et al. Aggregate features and adaboost for music classification. *Machine learning*, Springer, v. 65, n. 2-3, p. 473–484, 2006. Citado na página 17.

BINDER, R. V. *Testing object-oriented systems: models, patterns, and tools*. [S.l.]: Addison-Wesley Professional, 2000. Citado na página 34.

BOURQUE, P.; FAIRLEY, R. E. et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. [S.l.]: IEEE Computer Society Press, 2014. Citado na página 11.

BROWNE, M. W. Cross-validation methods. *Journal of mathematical psychology*, Elsevier, v. 44, n. 1, p. 108–132, 2000. Citado na página 71.

CALVI, A.; VIGANÒ, L. An automated approach for testing the security of web applications against chained attacks. In: ACM. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. [S.l.], 2016. p. 2095–2102. Citado 2 vezes nas páginas 13 e 97.

CHAN, W. et al. Reference models and automatic oracles for the testing of mesh simplification software for graphics rendering. In: IEEE. *Proceedings of the 30th Annual International Computer Software and Applications Conference, 2006. (COMPSAC 2006)*. [S.l.], 2006. v. 1, p. 429–438. Citado na página 101.

CHAN, W.; CHEUNG, S. C.; LEUNG, K. R. Towards a metamorphic testing methodology for service-oriented software applications. In: IEEE. *Fifth International Conference on Quality Software, 2005.(QSIC 2005)*. [S.l.], 2005. p. 470–476. Citado na página 45.

CHAN, W. K. et al. Pat: A pattern classification approach to automatic reference oracles for the testing of mesh simplification programs. *Journal of Systems and Software*, Elsevier, v. 82, n. 3, p. 422–434, 2009. Citado na página 100.

CHAVES, B. B. *Estudo do algoritmo AdaBoost de aprendizagem de máquina aplicado a sensores e sistemas embarcados*. Dissertação (Mestrado) — Universidade de São Paulo, São Paulo, 12 2011. Citado na página 17.

CHEN, J.; SUBRAMANIAM, S. Specification-based testing for gui-based applications. *Software Quality Journal*, Springer, v. 10, n. 3, p. 205–224, 2002. Citado na página 102.

CHEON, Y. Automated random testing to detect specification-code inconsistencies. 2007. Citado na página 101.

COHEN, J. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, Sage Publications, v. 20, n. 1, p. 37–46, 1960. Citado na página 25.

COHEN, W. W. Fast effective rule induction. In: *Twelfth International Conference on Machine Learning*. [S.l.]: Morgan Kaufmann, 1995. p. 115–123. Citado na página 16.

CUI, J. et al. A study of test oracle for application interface testing of distributed system. In: ATLANTIS PRESS. *2012 National Conference on Information Technology and Computer Science*. [S.l.], 2012. Citado na página 99.

DATTA, R. et al. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys (Csur)*, ACM, v. 40, n. 2, p. 5, 2008. Citado na página 33.

DAVIS, M. D.; WEYUKER, E. J. Pseudo-oracles for non-testable programs. In: ACM. *Proceedings of the ACM'81 Conference*. [S.l.], 1981. p. 254–257. Citado 2 vezes nas páginas 13 e 44.

DELAMARO, M. E.; NUNES, F. Lourdes dos S.; OLIVEIRA, R. A. P. Using concepts of content-based image retrieval to implement graphical testing oracles. *Software Testing, Verification and Reliability*, Wiley Online Library, v. 23, n. 3, p. 171–198, 2013. Citado 7 vezes nas páginas 5, 7, 27, 28, 42, 46 e 99.

DEYAB, H. H.; ATAN, R. B. Orchestration framework for automated ajax-based web application testing. In: IEEE. *Proceedings of the 9th Malaysian Software Engineering Conference, 2015. (MySEC 2015)*. [S.l.], 2015. p. 1–6. Citado 2 vezes nas páginas 13 e 97.

DING, J.; ZHANG, D. A machine learning approach for developing test oracles for testing scientific software. *the 28th SEKE (SEKE 2016), San Francisco, July*, p. 1–3, 2016. Citado na página 45.

ELYASOV, A. et al. Ab=ba: execution equivalence as a new type of testing oracle. In: ACM. *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. [S.l.], 2015. p. 1559–1566. Citado 2 vezes nas páginas 13 e 98.

FANG, L.; DOU, L.; XU, G. Perfblower: Quickly detecting memory-related performance problems via amplification. In: SCHLOSS DAGSTUHL-LEIBNIZ-ZENTRUM FUER INFORMATIK. *LIPIcs-Leibniz International Proceedings in Informatics*. [S.l.], 2015. v. 37. Citado na página 98.

FATTA, G. D.; LEUE, S.; STEGANTOVA, E. Discriminative pattern mining in software fault detection. In: ACM. *Proceedings of the 3rd international workshop on Software quality assurance*. [S.l.], 2006. p. 62–69. Citado 2 vezes nas páginas 42 e 101.

FRAIN, B. *Responsive Web Design with HTMl5 and CSS3*. [S.l.]: Packt Publishing Ltd, 2012. Citado na página 61.

FRASER, G.; ARCURI, A. Evosuite: automatic test suite generation for object-oriented software. In: ACM. *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. [S.l.], 2011. p. 416–419. Citado na página 100.

FRASER, G.; ARCURI, A. 1600 faults in 100 projects: automatically finding faults while achieving high coverage with evosuite. *Empirical software engineering*, Springer, v. 20, n. 3, p. 611–639, 2015. Citado na página 98.

FREUND, Y. An adaptive version of the boost by majority algorithm. *Machine learning*, Springer, v. 43, n. 3, p. 293–318, 2001. Citado na página 17.

FREUND, Y.; SCHAPIRE, R. E. A desicion-theoretic generalization of on-line learning and an application to boosting. In: SPRINGER. *European conference on computational learning theory*. [S.l.], 1995. p. 23–37. Citado 2 vezes nas páginas 16 e 17.

FREUND, Y.; SCHAPIRE, R. E. et al. Experiments with a new boosting algorithm. In: *icml*. [S.l.: s.n.], 1996. v. 96, p. 148–156. Citado 3 vezes nas páginas 16, 17 e 18.

FÜRNKRANZ, J.; WIDMER, G. Incremental reduced error pruning. In: MORGAN KAUFMANN. *Proceedings of the 11th International Conference on Machine Learning (ML-94)*. [S.l.], 1994. p. 70–77. Citado na página 16.

GAO, Z.; FANG, C.; MEMON, A. M. Pushing the limits on automation in gui regression testing. In: IEEE. *Proceedings of the IEEE 26th International Symposium on Software Reliability Engineering, 2015. (ISSRE 2015)*. [S.l.], 2015. p. 565–575. Citado 2 vezes nas páginas 13 e 97.

GAY, G. et al. Automated oracle data selection support. *IEEE Transactions on Software Engineering*, IEEE, v. 41, n. 11, p. 1119–1137, 2015. Citado na página 98.

GOSLING, J. *The Java language specification*. [S.l.]: Addison-Wesley Professional, 2000. Citado na página 61.

GUO, H.-F. A semantic approach for automated test oracle generation. *Computer Languages, Systems & Structures*, Elsevier, v. 45, p. 204–219, 2016. Citado na página 98.

GUO, H.-F. et al. Automated test oracle generation via denotational semantics. In: IEEE. *Proceedings of 2014 14th International Conference on Quality Software. (QSIC 2014)*. [S.l.], 2014. p. 139–144. Citado na página 98.

GUO, H.-F.; OUYANG, Q.; SIY, H. Semantics-based automated web testing. *arXiv preprint arXiv:1508.03905*, 2015. Citado na página 98.

HALL, M. et al. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, ACM, v. 11, n. 1, p. 10–18, 2009. Citado 2 vezes nas páginas 58 e 70.

HAMLET, D. Software quality, software process, and software testing. *Advances in Computers*, Elsevier, v. 41, p. 191–229, 1995. Citado na página 13.

HIGGINS, J. P.; GREEN, S. et al. *Cochrane handbook for systematic reviews of interventions*. [S.l.]: Wiley Online Library, 2008. v. 5. Citado na página 21.

HILLAH, L. M. et al. Service functional testing automation with intelligent scheduling and planning. In: ACM. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. [S.l.], 2016. p. 1605–1610. Citado 2 vezes nas páginas 13 e 97.

HOLT, N. E.; BRIAND, L. C.; TORKAR, R. Empirical evaluations on the cost-effectiveness of state-based testing: An industrial case study. *Information and Software Technology*, Elsevier, v. 56, n. 8, p. 890–910, 2014. Citado na página 98.

HORCH, J. W. *Practical guide to software quality management*. [S.l.]: Artech House, 2003. Citado na página 3.

HORI, A. et al. An oracle based on image comparison for regression testing of web applications. In: *SEKE*. [S.l.: s.n.], 2015. p. 639–645. Citado na página 98.

HOWDEN, W. E. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, SE-8, n. 4, p. 371–379, July 1982. ISSN 0098-5589. Citado na página 36.

HUNTER, C.; STROOPER, P. Systematically deriving partial oracles for testing concurrent programs. *Australian Computer Science Communications*, IEEE Computer Society Press, v. 23, n. 1, p. 83–91, 2001. Citado na página 102.

IEEE. Standard g lossary of softwareengineering terminology. *IEEE Software Engineering Standards & collection. IEEE*, p. 610–12, 1990. Citado na página 11.

IFTIKHAR, S. et al. An automated model based testing approach for platform games. In: IEEE. *Proceedings of the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, 2015. (MODELS 2015)*. [S.l.], 2015. p. 426–435. Citado na página 97.

JAHANGIROVA, G. et al. Oasis: oracle assessment and improvement tool. In: ACM. *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. [S.l.], 2018. p. 368–371. Citado na página 47.

JAMEEL, T.; MENGXIANG, L.; CHAO, L. Automatic test oracle for image processing applications using support vector machines. In: IEEE. *Proceedings of the 6th IEEE International Conference on Software Engineering and Service Science, 2015. (ICSESS 2015)*. [S.l.], 2015. p. 1110–1113. Citado na página 102.

JAMEEL, T.; MENGXIANG, L.; CHAO, L. A framework of automatic testing of image processing applications. In: IEEE. *Proceedings of the 13th International Bhurban Conference on Applied Sciences and Technology, 2016. (IBCAST 2016)*. [S.l.], 2016. p. 312–317. Citado 2 vezes nas páginas 13 e 97.

JIN, H. et al. Artificial neural network for automatic test oracles generation. In: IEEE. *Proceedings of the International Conference on Computer Science and Software Engineering, 2008*. [S.l.], 2008. v. 2, p. 727–730. Citado na página 101.

JIN, H. et al. Predication of program behaviours for functionality testing. In: IEEE. *Proceedings of the 1st International Conference on Information Science and Engineering, 2009. (ICISE 2009)*. [S.l.], 2009. p. 4993–4996. Citado na página 100.

KEELE, S. Guidelines for performing systematic literature reviews in software engineering. In: *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. [S.l.]: sn, 2007. Citado na página 21.

KIM-PARK, D. S.; RIVA, C. de la; TUYA, J. An automated test oracle for xml processing programs. In: ACM. *Proceedings of the First International Workshop on Software Test Output Validation*. [S.l.], 2010. p. 5–12. Citado na página 100.

KIRAÇ, M. F.; AKTEMUR, B.; SÖZER, H. Visor: A fast image processing pipeline with scaling and translation invariance for test oracle automation of visual output systems. *Journal of Systems and Software*, Elsevier, v. 136, p. 266–277, 2018. Citado na página 47.

KOHAVI, R. et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: MONTREAL, CANADA. *Ijcai*. [S.l.], 1995. v. 14, n. 2, p. 1137–1145. Citado na página 71.

KRUSE, P. M. Test oracles and test script generation in combinatorial testing. In: IEEE. *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops, 2016. (ICSTW 2016)*. [S.l.], 2016. p. 75–82. Citado na página 97.

KUHN, D. R.; OKUM, V. Pseudo-exhaustive testing for software. In: IEEE. *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop, 2006. (SEW 2006)*. [S.l.], 2006. p. 153–158. Citado na página 101.

LAMANCHA, B. P. et al. Automated generation of test oracles using a model-driven approach. *Information and Software Technology*, Elsevier, v. 55, n. 2, p. 301–319, 2013. Citado na página 99.

LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady*. [S.l.: s.n.], 1966. v. 10, n. 8, p. 707–710. Citado na página 23.

LIN, Y.-D. et al. On the accuracy, efficiency, and reusability of automated test oracles for android devices. *IEEE Transactions on Software Engineering*, IEEE, v. 40, n. 10, p. 957–970, 2014. Citado na página 98.

LIU, D. et al. Monic testing of web services based on algebraic specifications. In: IEEE. *Proceedings of the IEEE Symposium on Service-Oriented System Engineering, 2016. (SOSE 2016)*. [S.l.], 2016. p. 24–33. Citado 2 vezes nas páginas 13 e 97.

LUGER, G. F. *Artificial Intelligence: Structures and strategies for solving complex problems*. [S.l.]: Bookman, 2004. Citado na página 15.

MACHADO, P. D. Testing from structured algebraic specifications: The oracle problem. University of Edinburgh. College of Science and Engineering. School of Informatics., 2000. Citado na página 13.

MAJMA, N.; BABAMIR, S. M. Software test case generation & test oracle design using neural network. In: IEEE. *Proceedings of the 22nd Iranian Conference on Electrical Engineering, 2014. (ICEE 2014)*. [S.l.], 2014. p. 1168–1173. Citado na página 98.

MARIJAN, D. et al. Multimedia system verification through a usage model and a black test box. In: IEEE. *Proceedings of the International Conference on Computer Engineering and Systems, 2010. (ICCES 2010)*. [S.l.], 2010. p. 178–182. Citado na página 100.

MAYRHAUSER, A. von; ANDERSON, C.; MRAZ, R. Using a neural network to predict test case effectiveness. In: *Proceedings of IEEE Aerospace Applications Conference*. [S.l.: s.n.], 1995. p. 77–91 vol.2. Citado na página 4.

MCMASTER, S.; YUAN, X. Developing a feedback-driven automated testing tool for web applications. In: IEEE. *Proceedings of the 12th International Conference on Quality Software, 2012. (QSIC 2012)*. [S.l.], 2012. p. 210–213. Citado na página 99.

MEMON, A. et al. Dart: a framework for regression testing"nightly/daily builds"of gui applications. In: IEEE. *Proceedings of the International Conference on Software Maintenance, 2003. (ICSM 2003)*. [S.l.], 2003. p. 410–419. Citado na página 102.

MEMON, A.; NAGARAJAN, A.; XIE, Q. Automating regression testing for evolving gui software. *Journal of Software Maintenance and Evolution: Research and Practice*, Wiley Online Library, v. 17, n. 1, p. 27–64, 2005. Citado na página 101.

MEMON, A.; XIE, Q. Using transient/persistent errors to develop automated test oracles for event-driven software. In: IEEE. *Proceedings of the 19th International Conference on Automated Software Engineering, 2004*. [S.l.], 2004. p. 186–195. Citado na página 102.

MEMON, A. M. An event-flow model of gui-based applications for testing. *Software Testing Verification and Reliability*, Chichester, Sussex, England: J. Wiley, c1992-, v. 17, n. 3, p. 137–158, 2007. Citado na página 101.

MEMON, A. M.; POLLACK, M. E.; SOFFA, M. L. Automated test oracles for guis. In: ACM. *ACM SIGSOFT Software Engineering Notes*. [S.l.], 2000. v. 25, n. 6, p. 30–39. Citado na página 102.

MESBAH, A.; DEURSEN, A. V.; ROEST, D. Invariant-based automatic testing of modern web applications. *IEEE Transactions on Software Engineering*, IEEE, v. 38, n. 1, p. 35–53, 2012. Citado na página 99.

MEYER, B. et al. Automatic testing of object-oriented software. In: SPRINGER. *International Conference on Current Trends in Theory and Practice of Computer Science*. [S.l.], 2007. p. 114–129. Citado na página 101.

MICHAUD, R. O.; MICHAUD, R. O. *Efficient asset management: a practical guide to stock portfolio optimization and asset allocation*. [S.l.]: Oxford University Press, 2008. Citado na página 3.

MIRSHOKRAIE, S.; MESBAH, A.; PATTABIRAMAN, K. Pythia: Generating test cases with oracles for javascript applications. In: IEEE. *Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering, 2013. (ASE 2013)*. [S.l.], 2013. p. 610–615. Citado 2 vezes nas páginas 46 e 99.

MIRSHOKRAIE, S.; MESBAH, A.; PATTABIRAMAN, K. Atrina: Inferring unit oracles from gui test cases. In: IEEE. *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation, 2016. (ICST 2016)*. [S.l.], 2016. p. 330–340. Citado na página 97.

MOSIER, C. I. I. problems and designs of cross-validation 1. *Educational and Psychological Measurement*, Sage Publications Sage CA: Thousand Oaks, CA, v. 11, n. 1, p. 5–11, 1951. Citado na página 70.

MYERS, G. J.; SANDLER, C.; BADGETT, T. *The art of software testing*. [S.l.]: John Wiley & Sons, 2011. Citado 3 vezes nas páginas 3, 7 e 79.

NADEEM, A.; REHMAN, M. J.-U. Testaf: A test automation framework for class testing using object-oriented formal specifications. *J. UCS*, v. 11, n. 6, p. 962–985, 2005. Citado na página 101.

NARDO, D. D. et al. Model based test validation and oracles for data acquisition systems. In: IEEE. *Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering, 2013. (ASE 2013)*. [S.l.], 2013. p. 540–550. Citado na página 99.

NETO, A. C. D. Introduçao a teste de software. *Engenharia de Software Magazine*, v. 1, p. 22, 2007. Citado na página 12.

OLIVEIRA, R. A.; KANEWALA, U.; NARDI, P. A. Automated test oracles: State of the art, taxonomies, and trends. *Advances in Computers*, v. 95, p. 113–199, 2015. Citado 3 vezes nas páginas 13, 40 e 45.

OLIVEIRA, R. A. et al. An extensible framework to implement test oracle for non-testable programs. In: *SEKE*. [S.l.: s.n.], 2014. p. 199–204. Citado na página 98.

PACKEVIČIUS, Š.; UŠANIOV, A.; BAREIŠA, E. The use of model constraints as imprecise software test oracles. *Information technology and control*, v. 36, n. 2, 2015. Citado na página 102.

PADGHAM, L. et al. Model-based test oracle generation for automated unit testing of agent systems. *IEEE Transactions on Software Engineering*, IEEE, v. 39, n. 9, p. 1230–1244, 2013. Citado na página 99.

PÁDUA, W. de. *Engenharia de Software: Fundamentos, Métodos e Padrões*. [S.l.]: LTC,, 2008. Citado na página 12.

PASTORE, F.; MARIANI, L. Zoomin: discovering failures by detecting wrong assertions. In: IEEE PRESS. *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. [S.l.], 2015. p. 66–76. Citado na página 98.

PAVLIDIS, N. G. et al. Computational intelligence methods for financial time series modeling. *International Journal of Bifurcation and Chaos*, v. 16, n. 07, p. 2053–2062, 2006. Citado na página 4.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE 2008)*. [S.l.: s.n.], 2008. v. 8, p. 68–77. Citado na página 21.

PRESSMAN, R. S. *Software Engineering*. [S.l.]: Makron books Sao Paulo, 1995. v. 6. Citado 3 vezes nas páginas 3, 11 e 12.

RAN, L. et al. Building test cases and oracles to automate the testing of web database applications. *Information and Software Technology*, Elsevier, v. 51, n. 2, p. 460–477, 2009. Citado na página 100.

ROEST, D.; MESBAH, A.; DEURSEN, A. V. Regression testing ajax applications: Coping with dynamism. In: IEEE. *Proceedings of the Third International Conference on Software Testing, Verification and Validation, 2010. (ICST 2010)*. [S.l.], 2010. p. 127–136. Citado 2 vezes nas páginas 46 e 100.

SANGWAN, O. P.; BHATIA, P. K.; SINGH, Y. Radial basis function neural network based approach to test oracle. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 36, n. 5, p. 1–5, 2011. Citado na página 5.

SANTOS, R. L. d. S. et al. An experimental study based on fuzzy systems and artificial neural networks to estimate the importance of reviews about product and services. In: IEEE. *Neural Networks (IJCNN), 2016 International Joint Conference on*. [S.l.], 2016. p. 647–653. Citado na página 4.

SCHAPIRE, R. E.; SINGER, Y. Improved boosting algorithms using confidence-rated predictions. In: ACM. *Proceedings of the eleventh annual conference on Computational learning theory*. [S.l.], 1998. p. 80–91. Citado 2 vezes nas páginas 17 e 18.

SEIFERT, D. Conformance testing based on uml state machines: Automated test case generation, execution and evaluation. 2008. Citado na página 101.

SHAHAMIRI, S. R. et al. An automated framework for software test oracle. *Information and Software Technology*, Elsevier, v. 53, n. 7, p. 774–788, 2011. Citado na página 100.

SHAHAMIRI, S. R.; KADIR, W. M. N. W.; IBRAHIM, S. bin. An automated oracle approach to test decision-making structures. In: IEEE. *Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, 2010. (ICCSIT 2010)*. [S.l.], 2010. v. 5, p. 30–34. Citado na página 100.

SHAHAMIRI, S. R.; KADIR, W. M. W.; IBRAHIM, S. A single-network ann-based oracle to verify logical software modules. In: IEEE. *Proceedings of the 2nd International Conference on Software Technology and Engineering, 2010. (ICSTE 2010)*. [S.l.], 2010. v. 2, p. V2–272. Citado na página 100.

SHAHAMIRI, S. R. et al. Artificial neural networks as multi-networks automated test oracle. *Automated Software Engineering*, Springer, v. 19, n. 3, p. 303–334, 2012. Citado na página 99.

SHAW, M. Writing good software engineering research papers. In: IEEE. *Proceedings of the 25th International Conference on Software Engineering, 2003*. [S.l.], 2003. p. 726–736. Citado 2 vezes nas páginas 26 e 31.

SILVA, I. d.; SPATTI, D. H.; FLAUZINO, R. A. Artificial neural networks for engineering and applied sciences. *Sao Paulo: Artliber*, p. 33–111, 2010. Citado na página 15.

SIMON, H. A. Why should machines learn? In: *Machine learning*. [S.l.]: Springer, 1983. p. 25–37. Citado na página 15.

SINGHAL, A.; BANSAL, A.; KUMAR, A. An approach to design test oracle for aspect oriented software systems using soft computing approach. *International Journal of System Assurance Engineering and Management*, Springer, v. 7, n. 1, p. 1–5, 2016. Citado 2 vezes nas páginas 13 e 97.

SINGHAL, A.; BANSAL, A. et al. Generation of test oracles using neural network and decision tree model. In: IEEE. *Proceedings of the 5th International Conference on Confluence 2014: The Next Generation Information Technology Summit*. [S.l.], 2014. p. 313–318. Citado na página 98.

SOILA, P.; NARASIMHAN, P. *Causes of Failure in Web Applications*. [S.l.], 2005. Citado na página 3.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, Elsevier, v. 45, n. 4, p. 427–437, 2009. Citado 3 vezes nas páginas 18, 54 e 68.

SOUZA, M. de M. C. *Uma Abordagem para Apoiar Avaliações de Usabilidade de Sistemas Web Remotamente*. Dissertação (Mestrado) — Universidade Federal do Piauí, Teresina, 9 2016. Citado na página 6.

SPRENKLE, S. et al. Webvizor: A visualization tool for applying automated oracles and analyzing test results of web applications. In: IEEE. *Proceedings of the Testing: Academic & Industrial Conference Practice and Research Techniques, 2008. (TAIC PART 2008)*. [S.l.], 2008. p. 89–93. Citado na página 100.

SPRENKLE, S. et al. Automated replay and failure detection for web applications. In: ACM. *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. [S.l.], 2005. p. 253–262. Citado 2 vezes nas páginas 46 e 101.

SPRENKLE, S. et al. Automated oracle comparators for testingweb applications. In: IEEE. *Proceedings of the 18th IEEE International Symposium on Software Reliability, 2007. (ISSRE 2007)*. [S.l.], 2007. p. 117–126. Citado na página 101.

STAATS, M.; GAY, G.; HEIMDAHL, M. P. Automated oracle creation support, or: how i learned to stop worrying about fault propagation and love mutation testing. In: IEEE PRESS. *Proceedings of the 34th International Conference on Software Engineering*. [S.l.], 2012. p. 870–880. Citado na página 99.

SVENDSEN, A.; HAUGEN, Ø.; MØLLER-PEDERSEN, B. Specifying a testing oracle for train stations. In: ACM. *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation*. [S.l.], 2011. p. 5. Citado na página 99.

TAPPENDEN, A. F.; MILLER, J. Automated cookie collection testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 23, n. 1, p. 3, 2014. Citado 2 vezes nas páginas 42 e 102.

THORNTON, C. *Auto-WEKA: combined selection and hyperparameter optimization of supervised machine learning algorithms*. Tese (Doutorado) — University of British Columbia, 2014. Citado 2 vezes nas páginas 58 e 70.

TORSEL, A.-M. Automated test case generation for web applications from a domain specific model. In: IEEE. *Proceedings of the IEEE 35th Annual Computer Software and Applications Conference Workshops, 2011. (COMPSACW 2011)*. [S.l.], 2011. p. 137–142. Citado na página 99.

TRIPPI, R. R.; TURBAN, E. *Neural networks in finance and investing: Using artificial intelligence to improve real world performance*. [S.l.]: McGraw-Hill, Inc., 1992. Citado na página 3.

VANMALI, M.; LAST, M.; KANDEL, A. Using a neural network in the software testing process. *International Journal of Intelligent Systems*, Wiley Online Library, v. 17, n. 1, p. 45–62, 2002. Citado na página 102.

VEZHNEVETS, A.; VEZHNEVETS, V. Modest adaboost-teaching adaboost to generalize better. In: *Graphicon*. [S.l.: s.n.], 2005. v. 12, n. 5, p. 987–997. Citado na página 17.

WANG, F. et al. Evolving a test oracle in black-box testing. In: SPRINGER. *International Conference on Fundamental Approaches to Software Engineering*. [S.l.], 2011. p. 310–325. Citado na página 100.

WANG, F.; YAO, L.-W.; WU, J.-H. Intelligent test oracle construction for reactive systems without explicit specifications. In: IEEE. *Proceedings of the IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, 2011. (DASC 2011)*. [S.l.], 2011. p. 89–96. Citado na página 99.

WANG, X.; LI, Q. S. et al. An optimized method for automatic test oracle generation from real-time specification. In: IEEE. *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005. (ICECCS 2005)*. [S.l.], 2005. p. 440–449. Citado na página 101.

WANG, X.; WANG, J.; QI, Z.-C. Automatic generation of run-time test oracles for distributed real-time systems. In: SPRINGER. *International Conference on Formal Techniques for Networked and Distributed Systems*. [S.l.], 2004. p. 199–212. Citado na página 102.

WEYUKER, E. J. On testing non-testable programs. *The Computer Journal*, The British Computer Society, v. 25, n. 4, p. 465–470, 1982. Citado na página 13.

WIERINGA, R. et al. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, Springer, v. 11, n. 1, p. 102–107, 2006. Citado 2 vezes nas páginas 27 e 30.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012. Citado 3 vezes nas páginas 28, 39 e 77.

WU, J.; ZHANG, B.; WANG, K. Application of adaboost-based bp neural network for short-term wind speed forecast. *Power System Technology*, v. 36, n. 9, p. 221–225, 2012. Citado na página 17.

XIE, Q.; MEMON, A. M. Designing and comparing automated test oracles for gui-based software applications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 16, n. 1, p. 4, 2007. Citado na página 101.

XIE, T. Augmenting automatically generated unit-test suites with regression oracle checking. In: SPRINGER. *European Conference on Object-Oriented Programming*. [S.l.], 2006. p. 380–403. Citado na página 101.

XU, W. et al. Mining test oracles for test inputs generated from java bytecode. In: IEEE. *Proceedings of the IEEE 37th Annual Computer Software and Applications Conference, 2013. (COMPSAC 2013)*. [S.l.], 2013. p. 27–32. Citado na página 99.

XU, W.; WANG, H.; DING, T. Mining auto-generated test inputs for test oracle. In: IEEE. *Proceedings of the Tenth International Conference on Information Technology: New Generations, 2013. (ITNG 2013)*. [S.l.], 2013. p. 89–94. Citado na página 99.

XU, W.; XU, D. Automated evaluation of runtime object states against model-level states for state-based test execution. In: IEEE. *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops, 2009. (ICSTW 2009)*. [S.l.], 2009. p. 3–9. Citado na página 100.

YE, M. et al. Automated test oracle based on neural networks. In: IEEE. *Proceedings of the 5th IEEE International Conference on Cognitive Informatics, 2006. (ICCI 2006)*. [S.l.], 2006. v. 1, p. 517–522. Citado na página 101.

YOUSIF, M. E.; SHAHAMIRI, S. R.; MUSTAFA, M. B. Test oracles based on artificial neural networks and info fuzzy networks: A comparative study. In: IEEE. *Proceedings of the IEEE 10th Conference on Industrial Electronics and Applications, 2015. (ICIEA 2015)*. [S.l.], 2015. p. 467–471. Citado 2 vezes nas páginas 13 e 97.

ZAEEM, R. N.; PRASAD, M. R.; KHURSHID, S. Automated generation of oracles for testing user-interaction features of mobile apps. In: IEEE. *Proceedings of the IEEE Seventh International Conference on Software Testing, Verification and Validation, 2014. (ICST 2014)*. [S.l.], 2014. p. 183–192. Citado na página 98.

ZHAO, H. et al. Discovery of diagnosis pattern of coronary heart disease with qi deficiency syndrome by the t-test-based adaboost algorithm. *Evidence-Based Complementary and Alternative Medicine*, Hindawi Publishing Corporation, v. 2011, 2011. Citado na página 17.

Apêndices

APÊNDICE A – Referências para trabalhos selecionados ao Mapeamento Sistemático

Tabela 19 – Trabalhos selecionados no Mapeamento Sistemático de Estudos.

ID	Titulo	Referência
S01	Separating passing and failing test executions by clustering anomalies	(ALMAGHAIRBE; ROPER, 2016)
S02	Test Oracles and Test Script Generation in Combinatorial Testing	(KRUSE, 2016)
S03	Atrina: Inferring Unit Oracles from GUI Test Cases	(MIRSHOKRAIE; MESBAH; PATTABIRAMAN, 2016)
S04	Orchestration framework for automated Ajax-based web application testing	(DEYAB; ATAN, 2015)
S05	Monic testing of web services based on algebraic specifications	(LIU et al., 2016)
S06	An automated approach for testing the security of web applications against Chained Attacks	(CALVI; VIGANÒ, 2016)
S07	Service functional testing automation with intelligent scheduling and planning	(HILLAH et al., 2016)
S08	A framework of automatic testing of image processing applications	(JAMEEL; MENGXIANG; CHAO, 2016)
S09	An approach to design test oracle for aspect oriented software systems using soft computing approach	(SINGHAL; BANSAL; KUMAR, 2016)
S10	Pushing the limits on automation in GUI regression testing	(GAO; FANG; MEMON, 2015)
S11	An automated model based testing approach for platform games	(IFTIKHAR et al., 2015)
S12	Test oracles based on artificial neural networks and info fuzzy networks: A comparative study	(YOUSIF; SHAHAMIRI; MUSTAFA, 2015)
S13	On Proposing a Test Oracle Generator Based on Static and Dynamic Source Code Analysis	(ARANTES; SANTIAGO; VIJAYKUMAR, 2015)

S14	Automated Oracle Data Selection Support	(GAY et al., 2015)
S15	Semantics-based automated web testing	(GUO; OUYANG; SIY, 2015)
S16	ZoomIn: Discovering failures by detecting wrong assertions	(PASTORE; MARIANI, 2015)
S17	A semantic approach for automated test oracle generation	(GUO, 2016)
S18	Building Test Oracles by Clustering Failures	(ALMAGHAIRBE; ROPER, 2015)
S19	PERFBLOWER: Quickly detecting memory-related performance problems via amplification	(FANG; DOU; XU, 2015)
S20	1600 faults in 100 projects: automatically finding faults while achieving high coverage with EvoSuite	(FRASER; ARCURI, 2015)
S21	AB= BA: Execution equivalence as a new type of testing oracle	(ELYASOV et al., 2015)
S22	An oracle based on image comparison for regression testing of web applications	(HORI et al., 2015)
S23	On the accuracy, efficiency, and reusability of automated test oracles for android devices	(LIN et al., 2014)
S24	Automated generation of oracles for testing user-interaction features of mobile apps	(ZAEEM; PRASAD; KHURSHID, 2014)
S25	An extensible framework to implement test oracles for non-testable programs	(OLIVEIRA et al., 2014)
S26	Automated test oracle generation via denotational semantics	(GUO et al., 2014)
S27	Evaluating the TESTAR tool in an industrial case study	(BAUERSFELD et al., 2014)
S28	Generation of test oracles using neural network and decision tree model	(SINGHAL; BANSAL et al., 2014)
S29	Empirical evaluations on the cost-effectiveness of state-based testing: An industrial case study	(HOLT; BRIAND; TORKAR, 2014)
S30	Software test case generation & test oracle design using neural network	(MAJMA; BABAMIR, 2014)

S31	PYTHIA: Generating test cases with oracles for JavaScript applications	(MIRSHOKRAIE; MESBAH; PATTABIRAMAN, 2013)
S32	Model based test validation and oracles for data acquisition systems	(NARDO et al., 2013)
S33	Mining test oracles for test inputs generated from java bytecode	(XU et al., 2013)
S34	Mining auto-generated test inputs for test oracle	(XU; WANG; DING, 2013)
S35	Model-based test oracle generation for automated unit testing of agent systems	(PADGHAM et al., 2013)
S36	Using concepts of content-based image retrieval to implement graphical testing oracles	(DELAMARO; NUNES; OLIVEIRA, 2013)
S37	Automated generation of test oracles using a model-driven approach	(LAMANCHA et al., 2013)
S38	A study of test oracle for application interface testing of distributed system	(CUI et al., 2012)
S39	Developing a feedback-driven automated testing tool for web applications	(MCMASTER; YUAN, 2012)
S40	Artificial Neural Networks as multi-networks automated test oracle	(SHAHAMIRI et al., 2012)
S41	Automated oracle creation support, or: How I learned to stop worrying about fault propagation and love mutation testing	(STAATS; GAY; HEIMDAHL, 2012)
S42	A comparative study of artificial neural networks and info-fuzzy networks as automated oracles in software testing	(AGARWAL et al., 2012)
S43	Invariant-based automatic testing of modern web applications	(MESBAH; DEURSEN; ROEST, 2012)
S44	Intelligent test oracle construction for reactive systems without explicit specifications	(WANG; YAO; WU, 2011)
S45	Specifying a testing oracle for train stations	(SVENDSEN; HAUGEN; MØLLER-PEDERSEN, 2011)
S46	Automated test case generation for web applications from a domain specific model	(TORSEL, 2011)

S47	Semantic-based test oracles	(BAI et al., 2011)
S48	EvoSuite: Automatic test suite generation for object-oriented software	(FRASER; ARCURI, 2011)
S49	An automated framework for software test oracle	(SHAHAMIRI et al., 2011)
S50	Evolving a test oracle in black-box testing	(WANG et al., 2011)
S51	A single-network ANN-based oracle to verify logical software modules	(SHAHAMIRI; KADIR; IBRAHIM, 2010b)
S52	An automated test oracle for XML processing programs	(KIM-PARK; RIVA; TUYA, 2010)
S53	Multimedia system verification through a usage model and a black test box	(MARIJAN et al., 2010)
S54	An automated oracle approach to test decision-making structures	(SHAHAMIRI; KADIR; IBRAHIM, 2010a)
S55	A systematic capture and replay strategy for testing complex GUI based java applications	(ARISS et al., 2010)
S56	Regression testing Ajax applications: Coping with dynamism	(ROEST; MESBAH; DEURSEN, 2010)
S57	Predication of program behaviours for functionality testing	(JIN et al., 2009)
S58	Utilizing an abstraction relation document in grey-box testing approach	(BAHAROM; SHUKUR, 2009)
S59	Automated evaluation of runtime object states against model-level states for state-based test execution	(XU; XU, 2009)
S60	PAT: A pattern classification approach to automatic reference oracles for the testing of mesh simplification programs	(CHAN et al., 2009)
S61	Building test cases and oracles to automate the testing of web database applications	(RAN et al., 2009)
S62	WebVizOR: A visualization tool for applying automated oracles and analyzing test results of web applications	(SPRENKLE et al., 2008)

S63	Conformance testing based on UML state machines: Automated test case generation, execution and evaluation	(SEIFERT, 2008)
S64	Artificial neural network for automatic test oracles generation	(JIN et al., 2008)
S65	Automated random testing to detect specification-code inconsistencies	(CHEON, 2007)
S66	Automated oracle comparators for testing web applications	(SPRENKLE et al., 2007)
S67	Automatic testing of object-oriented software	(MEYER et al., 2007)
S68	An event-flow model of GUI-based applications for testing	(MEMON, 2007)
S69	Designing and comparing automated test oracles for GUI-based software applications	(XIE; MEMON, 2007)
S70	Reference models and automatic oracles for the testing of Mesh simplification software for graphics rendering	(CHAN et al., 2006)
S71	Discriminative pattern mining in software fault detection	(FATTA; LEUE; STEGANTOVA, 2006)
S72	Pseudo-exhaustive testing for software	(KUHN; OKUM, 2006)
S73	Automated test oracle based on neural networks	(YE et al., 2006)
S74	Augmenting automatically generated unit-test suites with regression oracle checking	(XIE, 2006)
S75	Automated replay and failure detection for web applications	(SPRENKLE et al., 2005)
S76	An optimized method for automatic test oracle generation from real-time specification	(WANG; LI et al., 2005)
S77	TESTAF: A test automation framework for class testing using object-oriented formal specifications	(NADEEM; REHMAN, 2005)
S78	Automating regression testing for evolving GUI software	(MEMON; NAGARAJAN; XIE, 2005)

S79	Automatic generation of run-time test oracles for distributed real-time systems	(WANG; WANG; QI, 2004)
S80	Dart: A Framework for Regression Testing "Nightly/daily Builds"of GUI Applications	(MEMON et al., 2003)
S81	Specification-based Testing for GUI-based Applications	(CHEN; SUBRAMANIAM, 2002)
S82	Using a neural network in the software testing process	(VANMALI; LAST; KANDEL, 2002)
S83	Systematically deriving partial oracles for testing concurrent programs	(HUNTER; STROOPER, 2001)
S84	Automated test oracles for GUIs	(MEMON; POLLACK; SOFFA, 2000)
S85	Automatic Test Oracle for Image Processing Applications Using Support Vector Machines	(JAMEEL; MENGXIANG; CHAO, 2015)
S86	Automated Cookie Collection Testing	(TAPPENDEN; MILLER, 2014)
S87	The use of model constraints as imprecise software test oracles	(PACKEVIČIUS; UŠANIOV; BAREIŠA, 2015)
S88	Using transient/persistent errors to develop automated test oracles for event-driven softwareCA	(MEMON; XIE, 2004)