UNIVERSIDADE FEDERAL DE PERNAMBUCO

Centro de Informática

Ciência da Computação

Teste de Software com IA: Um Mapeamento Sistemático da Literatura

Jailson da Costa Dias

Recife

Abril de 2023

Teste de Software com IA: Um Mapeamento Sistemático da Literatura

Jailson da Costa Dias

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Juliano Manabu Iyoda

Recife

Abril de 2023

Teste de Software com IA: Um Mapeamento Sistemático da Literatura

Jailson da Costa Dias

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Data da Defesa do TCC:

Recife, 20 de Abril de 2022

BANCA EXAMINADORA

_____

Prof. Juliano Manabu Iyoda (Orientador)

UNIVERSIDADE FEDERAL DE PERNAMBUCO

_____

Prof. Breno Alexandro Ferreira de Miranda (2º membro da banca)

UNIVERSIDADE FEDERAL DE PERNAMBUCO

Agradecimentos

Gostaria de expressar minha profunda gratidão aos meus pais e irmãos por todo o amor e incentivo que me deram ao longo dos anos. Sem o apoio deles, eu não teria chegado onde estou hoje e não seria a pessoa que sou. Amo vocês profundamente.

Também gostaria de agradecer a todos os meus amigos, especialmente aqueles que estiveram ao meu lado durante a jornada da graduação. Obrigado por tornarem todas as noites de trabalho no CIn mais leves e divertidas com sua companhia e amizade.

Um agradecimento especial ao meu orientador, Juliano, por todo o apoio e orientação. Sou grato pelo tempo que dedicou a me orientar e pelo valor inestimável que isso teve para mim.

Agradeço aos membros da banca por aceitarem avaliar este trabalho e a todos aqueles que contribuíram de alguma forma para minha formação pessoal e acadêmica.

Muito obrigado a todos vocês por tudo. Vocês fizeram uma grande diferença na minha vida.

Resumo

As técnicas de teste de software são essenciais para detectar falhas e mitigar os riscos associados ao desenvolvimento de software. Esse processo fica ainda mais eficiente com o uso de Inteligência Artificial (IA). Como resultado, é fundamental entender essas estratégias e como aplicá-las no processo de desenvolvimento de software.

Para atingir esse objetivo, um mapeamento sistemático da literatura foi realizado para fornecer uma visão geral do estado da arte em técnicas de teste de software utilizando IA e revelar a atual distribuição de pesquisas nessa área. Neste mapeamento sistemático, 191 estudos primários foram sistematicamente coletados e categorizados com base nas técnicas de teste de software pesquisadas e nos algoritmos ou técnicas de IA utilizadas.

Isso permitiu a identificação da distribuição da pesquisa por categoria de técnicas de teste de software estudadas, os algoritmos de IA mais comumente utilizados nesses estudos e a identificação de lacunas na pesquisa. Além disso, resultou em um mapeamento das categorias com artigos relevantes, permitindo aos pesquisadores localizar rapidamente todas as pesquisas sobre técnicas de teste de software utilizando IA com as propriedades de seu interesse.

Palavras-chave: Teste de Software, Inteligência Artificial, Mapeamento Sistemático

Abstract

Software testing techniques are essential for detecting failures and mitigating the risks associated with software development. This process is made even more efficient with the use of Artificial Intelligence (AI). As a result, it is critical to understand these strategies and how to apply them in the software development process.

To achieve this goal, a systematic mapping study was performed to provide an overview of the state of the art in software testing techniques using AI and reveal the current distribution of research in this area. In this systematic mapping, 191 primary studies were systematically collected and categorized based on the software testing techniques researched and the algorithms or AI techniques used.

This allowed for the identification of the research distribution by category of software testing techniques studied, the most commonly used AI algorithms in these studies, and the identification of gaps in research. In addition, it resulted in a mapping of categories with relevant articles, allowing researchers to quickly locate all research on software testing techniques using AI with properties of their interest.

Keywords: Software Testing, Artificial Intelligence, Systematic Mapping

Lista de Abreviaturas e Siglas

IA	Inteligência Artificial
TCP	Priorização de Casos de Teste
TCS	Seleção de Casos de Teste
TCG	Geração de Casos de Teste
TDG	Geração de Dados de Teste
TSG	Geração de Suíte de Teste
TSR	Redução da Suíte de Teste
TSP	Predição de Suíte de Teste
RNN	Rede Neural Recorrente
CNN	Rede Neural Convolucional
LSTM	Memória de Longo Prazo de Curto Prazo
GNN	Rede Grafos Neurais
DRL	Aprendizado por Reforço Profundo
DNN	Rede Neural Profunda
GA	Algoritmo Genético
SVM	Máquina de Vetores de Suporte
HGA	Algoritmo Genético Híbrido
WOA	Algoritmo de Otimização de Baleias
NLP	Processamento de Linguagem Natural
GCN	Rede de Grafos Convolutivos
GGA	Algoritmo Genético Guiado
GAN	Rede Adversária Generativa
KNN	K-vizinhos mais próximos
MLP	Multilayer perceptron

SA	Simulated Annealing
AHC	Agrupamento Hierárquico
ACA	Algoritmo de Agrupamento Aglomerativo
TCN	Redes Convolucionais Temporais
FPA	Algoritmo de Polinização de Flores
ACO	Otimização por Colônia de Formigas
GIN	Redes de Isomorfismo em grafos
TCA	Análise de Componentes de Transferência
PSO	Algoritmo de Otimização por Enxame
RAR	Random Approximate Reduct
PS	Prioritized Sweeping
TD	Temporal Difference
DE	Differential Evolution
ABC	Colônia de Abelhas Artificial
ARA	Algoritmo de Reprodução Assexuada
AFSA	Algoritmo de Cardume de Peixes Artificial
GPE	Algoritmo de Evolução de Previsão Cinza
POMCP	Planejamento de Monte Carlo Parcialmente Observável

Sumário

1. Introdução

Com o aumento da complexidade dos softwares desenvolvidos e o aumento da competitividade, é vital elevar a qualidade do software desenvolvido a novos patamares. Como isso, o teste de software torna-se uma etapa crítica no desenvolvimento do sistema para verificar a qualidade e a confiabilidade do produto final.

Os testes de software permitem detectar e corrigir erros e falhas que possam comprometer o bom funcionamento do sistema. Além disso, os testes do software podem auxiliar na identificação de problemas de desempenho, segurança e usabilidade, entre outras coisas. Assim, a execução de testes de software é fundamental para garantir a confiabilidade do sistema desenvolvido e a satisfação dos usuários, além de evitar prejuízos financeiros e de imagem para a empresa desenvolvedora do software.

Com a evolução da Inteligência Artificial (IA), novas técnicas e algoritmos para auxiliar no processo de teste de software foram desenvolvidos. A IA tem o potencial de revolucionar a maneira de como o teste de software é realizado, automatizando muitos dos processos envolvidos no teste do software, utilizando técnicas como aprendizado de máquina e processamento de linguagem natural. Isso pode reduzir significativamente o tempo e o esforço necessários para realizar testes completos e precisos.

Neste trabalho, temos como objetivo fazer um mapeamento sistemático da literatura sobre técnicas de teste de software utilizando IA. Esperamos que, ao fazer isso, os pesquisadores possam identificar facilmente todas as pesquisas sobre técnicas de teste de software que utilizam IA e que contenham as propriedades de seu interesse.

1.1. Objetivo

Um estudo de mapeamento sistemático é uma metodologia útil para fornecer uma visão geral de uma área de pesquisa, classificando os artigos nela e contando

o número de artigos pertencentes a cada categoria na classificação. Por exemplo, é possível classificar os artigos publicados em um campo de pesquisa por seu ano de publicação, com cada ano representando uma categoria. Neste caso, contando o número de artigos publicados em cada categoria e em cada ano, pode nos dar uma ideia do nível de atividade no campo ao longo do tempo. Da mesma forma, classificar os artigos com base em seu conteúdo nos dá uma noção do que é comumente pesquisado e onde há lacunas de pesquisa. Tais classificações também podem fornecer informações de alto nível sobre o estado atual da arte. Como um exemplo, classificar os artigos pelas técnicas de inteligência artificial (IA) utilizadas e as técnicas de teste de software estudadas, dá uma noção geral de quais testes de softwares são comumente estudados e quais técnicas de IA são mais utilizadas nesses estudos.

Considerando essas classificações no contexto de testes de software utilizando IA, como a técnica de IA utilizada e a técnica de teste de software estudada, podemos responder a perguntas mais interessantes sobre o estado da arte: Qual técnica de IA é mais comumente utilizada para automação da priorização de casos de testes? Qual é a distribuição das técnicas de IA utilizadas para a seleção de casos de testes? Além disso, as classificações podem ser usadas para construir um mapeamento de categorias para conjuntos de artigos pertencentes a elas; permitindo que os pesquisadores localizem facilmente artigos no campo pertencentes às categorias em que estão interessados. Aqui, utilizamos um mapeamento sistemático da literatura no campo de pesquisa de testes de software utilizando IA para alcançar nossos objetivos principais de resumir as tendências recentes de publicação e identificar como a IA está sendo aplicada nos testes de software.

2. Fundamentos

2.1. Teste de software

O processo de Teste de Software é uma etapa muito importante no fluxo de desenvolvimento de software, pois nele vai ser verificada e validada a qualidade do software desenvolvido, assim como o comportamento e desempenho do mesmo, a

fim de garantir que ele atenda aos requisitos de negócios, usabilidade, segurança, infraestrutura entre outros [6].

A origem do Teste de Software remonta aos primórdios do desenvolvimento de software, com o cientista da computação Tom Kilburn, que é creditado por escrever o primeiro programa de computador com armazenamento em 1948 na Universidade de Manchester, na Inglaterra [7]. Esse software realizava cálculos matemáticos usando instruções de código de máquina. Já Joseph Juran é creditado como o pai do teste de software que, pela primeira vez, em 1951, marcou a importância da garantia de qualidade de software em seu livro "Quality Control Handbook" [7, 8]. Desde então, Teste de Software tem evoluído junto com o desenvolvimento de software, como demonstrado por Joris Meerts e Dorothy Graham em "*The History of Software Testing*" [9].

2.2. Inteligência Artificial

A Inteligência Artificial (IA) é um campo de pesquisa que tem como objetivo desenvolver dispositivos capazes de realizar tarefas que normalmente requerem inteligência humana, como reconhecimento de padrões, aquisição de conhecimento e tomada de decisões [10].

Segundo Smith *et al.* [11], o termo "Inteligência Artificial" foi cunhado pela primeira vez por John McCarthy em 1956, quando realizou a primeira conferência acadêmica sobre o assunto. Mas o estudo da Inteligência Artificial começou antes, com o trabalho de Vannevar Bush, "As We May Think", que propôs um sistema que amplificaria o conhecimento e a compreensão das pessoas e com o trabalho de Alan Turing, em 1950, o qual escreveu um artigo sobre a noção de máquinas serem capazes de simular seres humanos e fazer coisas inteligentes, como jogar xadrez [11]. Desde então, a IA tem evoluído rapidamente e tem sido aplicada em diversas áreas, incluindo jogos, reconhecimento de fala e imagem, diagnóstico médico e veículos autônomos.

2.3. Mapeamento sistemática da literatura

Segundo Petersen *et al.* [13], a revisão sistemática da literatura busca fornecer uma visão geral de uma área de pesquisa, estruturando os relatórios e resultados de pesquisa que foram publicados, categorizando-os e geralmente fornecendo um resumo visual, o mapa, dos seus resultados. Ainda de acordo com Petersen *et al.* [13], além da visão geral da literatura, muitas vezes também deseja-se mapear as frequências de publicação ao longo do tempo para identificar tendências. A Figura 1 apresenta os principais passos do processo de mapeamento sistemático sugeridos por Petersen *et al*. [13].

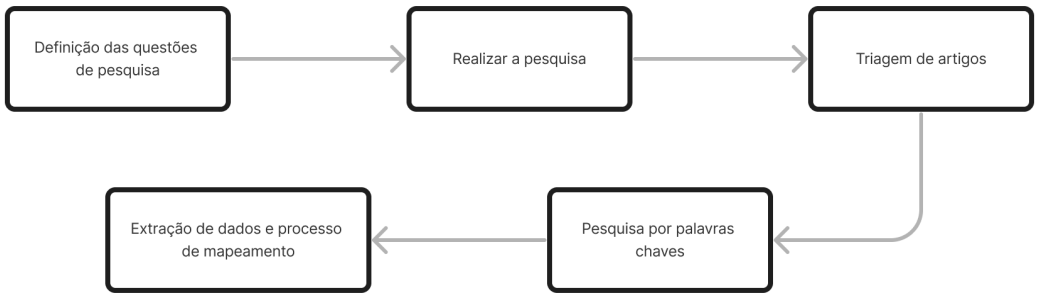


Figura 1. Processo de mapeamento sistemático de Petersen *et al.*

Como podemos observar na figura 1, para realizar um mapeamento sistemático da literatura, é necessário definir as questões de pesquisa e conduzir uma busca nas bases de dados relevantes, utilizando palavras-chave. Depois, faz a triagem para selecionar os artigos mais pertinentes e uma busca por palavras chaves utilizando o abstract para selecionar os artigos mais relevantes. Por fim, faz a extração dos dados e mapeamento dos dados extraídos para verificar possíveis relações e padrões existentes.

3. Metodologia

Conforme sugerido por Petersen *et al.* [13], iniciamos o processo de mapeamento sistemático da literatura definindo as questões de pesquisa de acordo

com os objetivos estabelecidos. Em seguida, realizamos uma busca sistemática, coletando um conjunto de artigos representativos do campo de interesse. Depois disso, selecionamos os estudos mais relevantes de acordo com os objetivos estabelecidos e mapeamos os dados em categorias para estruturar o campo e fornecer respostas às questões de pesquisa previamente estabelecidas.

3.1. Questões de pesquisa

Nós começamos derivando as questões de pesquisa a partir dos objetivos principais deste trabalho. Como afirmado acima, gostaríamos de estruturar o campo de pesquisa de teste de software utilizando IA e identificar como a IA está sendo aplicada no fluxo de testes de software, analisando os artigos do campo de estudo.

As seguintes perguntas são derivadas dos objetivos.

1. Quais são as tendências de publicação em pesquisas de testes de software utilizando inteligência artificial?

    1.1. Qual é o número anual de publicações na área de teste de software utilizando IA?

    1.2. Quais são as principais técnicas de testes de software estudadas?

    1.3. Quais são as principais técnicas de inteligência artificial utilizadas?

3.2. *String* de busca

Nossa *string* de busca foi criada primeiro dividindo os fenômenos em estudo em termos principais. Para cada termo principal, foram adicionadas palavras-chave sinônimas usando o operador OR. As palavras-chave foram escolhidas com base nas nossas questões de pesquisa e no escopo do objetivo da pesquisa. Os termos principais foram unidos usando o operador AND. A *string* de busca resultante foi aprimorada de forma iterativa, avaliando sua capacidade de gerar artigos relevantes a partir de pequenos subconjuntos de artigos e ajustando as palavras-chave conforme necessário. Chegamos à seguinte *string* de busca:

("software testing" OR "black-box testing" OR "white-box testing") AND ("AI" OR "artificial intelligence" OR "machine learning" OR "ML" OR "deep learning" OR "Genetic Algorithm")

Para as bases de dados científicas, escolhemos algumas das fontes online mais populares: ACM, IEEE Xplore e ScienceDirect.

Devido ao nosso amplo escopo e interesse no estado atual e lacunas de pesquisa, mas com restrições fortes de tempo, limitamos nossa busca para incluir apenas artigos publicados nos últimos 3 anos [2020-2022]. Também excluímos livros dos nossos resultados de busca, pois estamos interessados em trabalhos acadêmicos revisados por pares com maior probabilidade de serem de alta qualidade. Aplicamos nossa *string* de busca em cada um dos bancos de dados online para obter 8.902 artigos potencialmente relevantes.

3.3. Estratégia de pesquisa

Para selecionar artigos relevantes para o mapeamento sistemático da literatura sobre testes de software utilizando IA, foi necessário seguir um conjunto de etapas, que incluem a aplicação dos seguintes critérios:

1. Critérios de inclusão, que determinam quais artigos são relevantes para o tema e devem ser incluídos no estudo;
2. Critérios de exclusão, que determinam quais artigos não são relevantes para o tema e devem ser excluídos do estudo; e
3. Critérios de qualidade, que avaliam a qualidade dos artigos selecionados e removem os artigos que não atendem aos critérios de qualidade estabelecidos.

A aplicação desses critérios é essencial para garantir que apenas artigos relevantes e de alta qualidade sejam incluídos na análise final do mapeamento sistemático da literatura.

3.3.1. Critérios de inclusão

Os critérios de inclusão para a seleção de artigos foram estabelecidos para garantir a relevância e qualidade dos artigos selecionados. Primeiramente, foi considerada a acessibilidade dos artigos, incluindo apenas aqueles disponíveis gratuitamente por meio da *Virtual Private Network* (VPN) institucional do Centro de Informática da UFPE. Além disso, para garantir a atualidade dos artigos incluídos e que eles retratam o estado atual do campo de pesquisa, foram incluídos apenas artigos publicados entre os anos de 2020 e 2022.

Para evitar a fuga ao tema, foram selecionados apenas artigos relacionados diretamente com o tópico abordado, incluindo estudos primários que passaram por revisão dos pares antes de sua publicação. Na Tabela 1 abaixo é possível observar os critérios de inclusão (CI_n).

Tabela 1 - Critérios de Inclusão dos artigos

Critério	**Descrição**
CI1	Publicações disponíveis através da VPN do centro de informática - UFPE
CI2	Trabalhos publicados entre 2020 e 2022
CI3	Publicações sobre Técnicas de Teste de Software
CI4	Estudos primários
CI5	Artigos revisados por pares
CI6	Ter técnicas específicas de IA

Fonte: O autor

3.3.2. Critérios de exclusão

Os critérios de exclusão foram estabelecidos para garantir que apenas artigos relevantes e de alta qualidade fossem incluídos no estudo. Para isso, foram

estabelecidos critérios como a exclusão de artigos pagos, revisões sistemáticas da literatura (RSLs), publicações duplicadas e aquelas sem revisão por pares.

Além disso, foram excluídos artigos indisponíveis para download ou visualização. Na Tabela 2 abaixo é possível observar os critérios de exclusão (CE).

Tabela 2 - Critérios de Exclusão dos artigos

Critério	Descrição
CE1	Artigos com conteúdo pago
CE2	Artigos secundários (revisão ou mapeamento da literatura)
CE3	Artigos duplicados
CE4	Artigos indisponíveis para download ou visualização
CE5	Publicações sem revisão por pares
CE6	Não ter técnicas específicas de IA

Fonte: O autor

3.3.3. Critérios de qualidade

Para avaliar a qualidade dos artigos selecionados, foi aplicado um terceiro filtro, que consistiu em realizar uma leitura diagonal do conteúdo dos artigos. Esse processo teve como objetivo avaliar a qualidade do conteúdo dos artigos, identificar as técnicas de IA utilizadas e verificar se a técnica de teste de software estudada foi especificada. Além disso, foram considerados critérios como a completude e clareza do conteúdo apresentado. Os artigos que não atenderam a esses critérios foram eliminados do estudo. Na Tabela 3 abaixo é possível observar os critérios de qualidade (CQ_n).

Tabela 3 - Critérios de Qualidade dos artigos

Critério	Descrição
CQ1	A técnica de IA utilizada foi especificada?

CQ2	A técnica de teste de software estudada foi especificada?
CQ3	Os objetivos do artigo são apresentados de forma clara?
CQ4	Completude e clareza do conteúdo

Fonte: O autor

3.4. Filtros desenvolvidos

Com os critérios de inclusão, exclusão e qualidade definidos, a próxima fase no processo de seleção dos artigos para o estudo consiste em desenvolver filtros para garantir a seleção dos artigos adequados para a condução do trabalho. O primeiro filtro foi baseado no ano de publicação dos artigos e selecionou apenas aqueles publicados entre 2020 e 2022. O segundo e terceiro filtros foram baseados na leitura do título e do resumo (abstract) dos artigos, respectivamente, e buscaram selecionar apenas os artigos que estavam diretamente relacionados com o tema abordado. No quarto filtro, foi realizada uma leitura diagonal, a qual teve como objetivo avaliar mais especificamente a qualidade dos artigos, levando em consideração a presença de informações importantes como as técnicas de IA utilizadas e as técnicas de teste de software estudadas, bem como a clareza e completude do conteúdo. Na Tabela 4 abaixo é possível observar os filtros (F_n) desenvolvidos para a seleção de artigos finais do mapeamento sistemático da literatura.

Tabela 4 - Filtro para a seleção de artigos

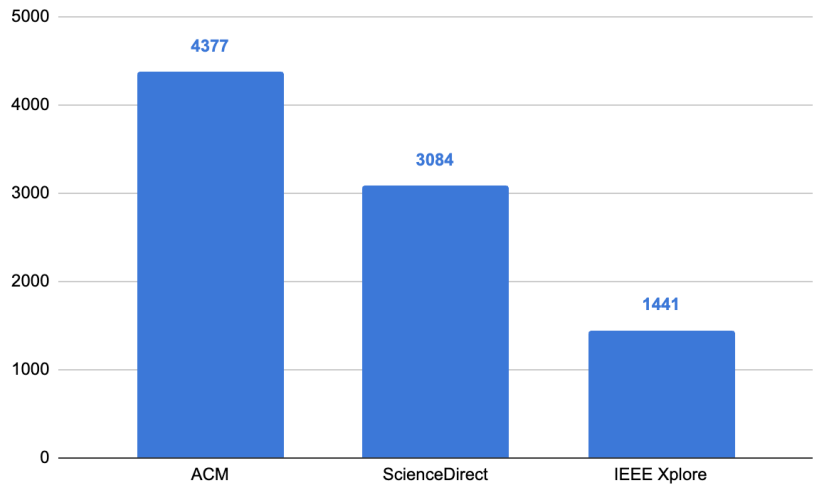
Critério	**Descrição**
F1	Ano de publicação
F2	Leitura do título
F3	Leitura do resumo (*abstract*)
F4	Leitura diagonal para avaliação da qualidade do artigo

Fonte: O autor

3.5. Seleção de artigos

No início do processo de seleção de artigos para o estudo, havia uma grande quantidade de artigos disponíveis para análise. O número total de artigos era de 8.902, o que representa uma ampla base de informações para consulta. Esses artigos estavam divididos entre as bases de dados da IEEE Xplore, ACM e ScienceDirect, conforme apresentado no Gráfico 1 abaixo.

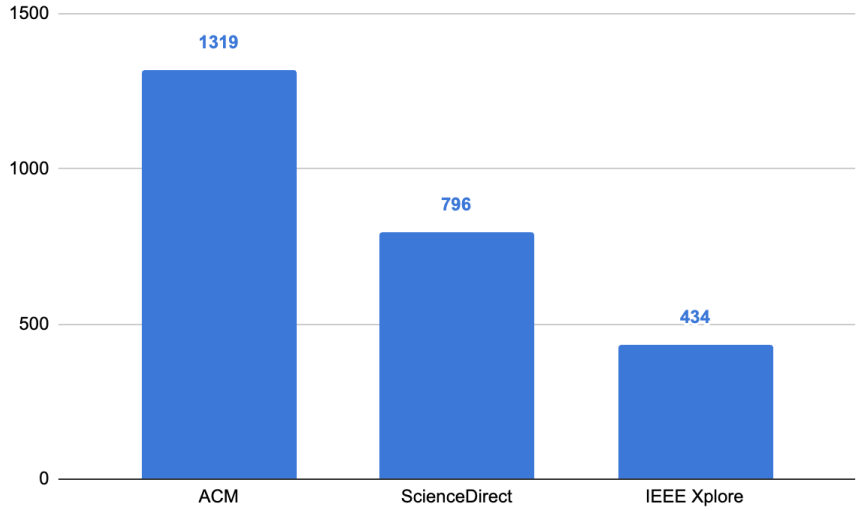
Gráfico 1 - Artigos retornado pela string de busca



Fonte: O autor

Devido ao grande número de artigos retornados na *string* de busca e ao nosso objetivo de encontrar tendências de teste de software nos últimos anos, foi aplicado o filtro dos artigos pelo ano de publicação (F1) diretamente nas plataformas das base de dados, resultando em 2.549 artigos para análise dos anos de 2020 a 2022. Esses artigos estavam divididos entre as bases de dados da IEEE Xplore, ACM e ScienceDirect, conforme apresentado no Gráfico 2.

Gráfico 2 - Artigos depois do filtro F1

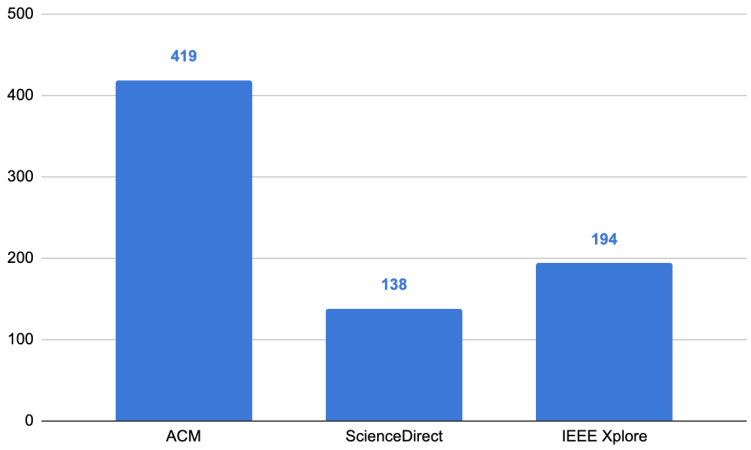


Fonte: O autor

Após o filtro pelo ano de publicação, o próximo filtro aplicado foi a leitura dos títulos dos artigos (F2). Esse filtro teve como objetivo eliminar os artigos que não apresentavam relação com a área de estudo do trabalho, utilizando os critérios de inclusão e exclusão definidos. Após a aplicação desse filtro, restaram 751 artigos nas bases de dados da IEEE Xplore, ACM e ScienceDirect. O Gráfico 3 apresenta a distribuição desses artigos nas bases de dados.

Essa etapa de leitura dos títulos foi feita por Jailson da Costa Dias e Juliano Manabu Iyoda, onde a maior contribuição foi feita por Jailson da Costa Dias. Juliano Manabu Iyoda leu e selecionou apenas uma ínfima quantidade desses artigos.

Gráfico 3 - Artigos depois do filtro F2

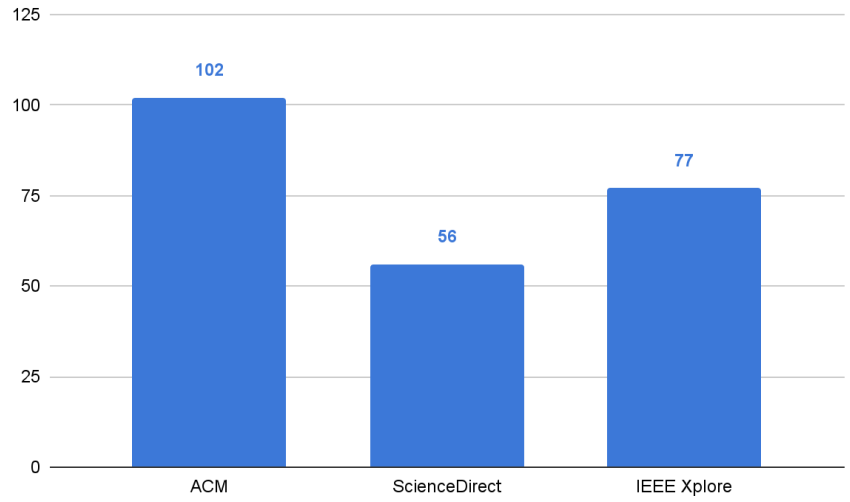


Fonte: O autor

Em seguida, foi realizado o filtro da leitura dos *abstracts* dos artigos (F3). Assim como o filtro anterior, esse filtro teve como objetivo eliminar os artigos que não apresentavam relação com a área de estudo do trabalho, utilizando os critérios de inclusão e exclusão definidos. Como tinham muitos *abstracts* para serem lidos, nessa etapa foi utilizada uma heurística para eliminar os artigos que não mencionaram o uso de inteligência artificial em seu conteúdo. Para fazer isso, foram utilizadas as palavras chaves *learn, network, processing, computer, fuzzy, genetic, algorithm, intelligence, predictive, agent, artificial, cluster, classification, metaheuristic* e *evolutionary*, e foram eliminados todos os artigos que não mencionaram nenhuma dessas palavras chaves em seu *abstract*, título ou palavras-chave. Após a aplicação desse filtro, restaram 235 artigos nas bases de dados da IEEE Xplore, ACM e ScienceDirect. O Gráfico 4 apresenta a distribuição desses artigos nas bases de dados.

Já na etapa de leitura dos *abstracts*, Jailson da Costa Dias realizou todo o processo de eliminação de artigos utilizando a heurística definida e leu o abstract dos artigos restantes.

Gráfico 4 - Artigos depois do filtro F3

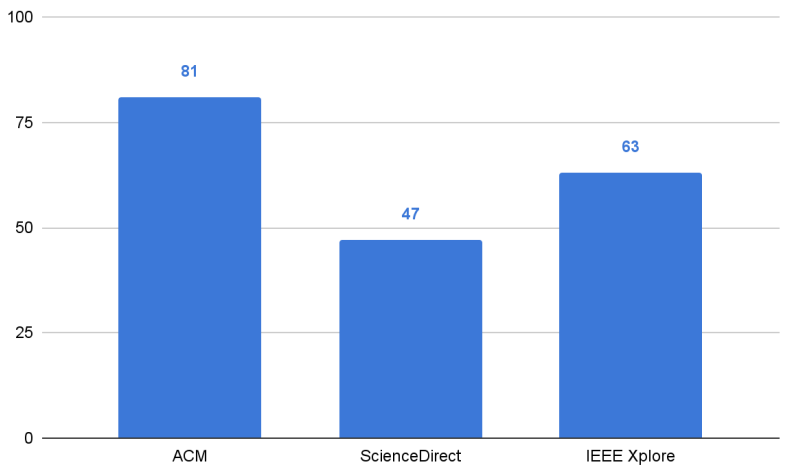


Fonte: O autor

Por fim, foi realizado o filtro da leitura diagonal dos artigos restantes (F4). Esse filtro teve como objetivo eliminar os artigos que não apresentavam boa qualidade, de acordo com os critérios de qualidade estabelecidos. Após a aplicação desse filtro, restaram 191 artigos nas bases de dados da IEEE Xplore, ACM e ScienceDirect. O Gráfico 5 apresenta a distribuição desses artigos nas bases de dados.

Nessa última etapa, de leitura diagonal, Jailson da Costa Dias realizou a leitura diagonal de todos os artigos 235 artigos restantes após a aplicação do filtro F3.

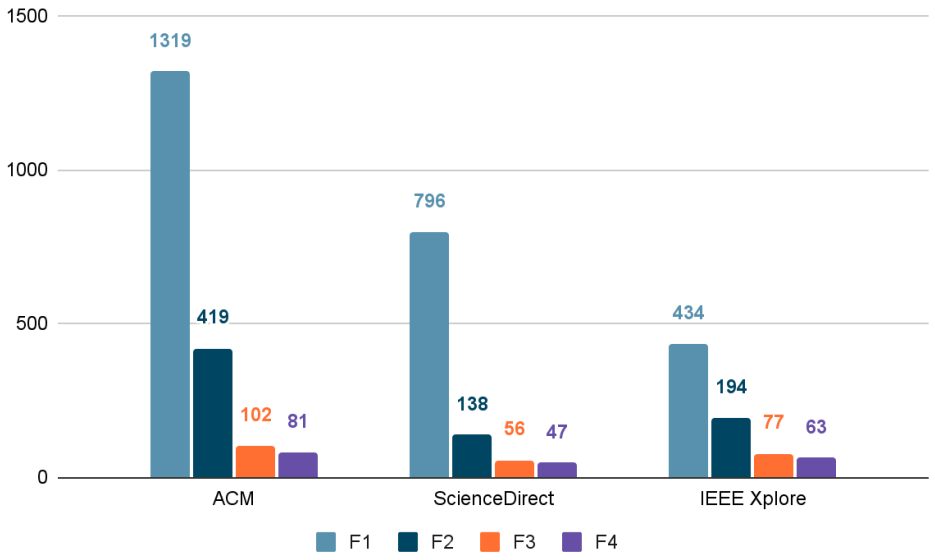
Gráfico 5 - Artigos depois do filtro F4



Fonte: O autor

Depois de passar por todos os filtros, abaixo está apresentado o Gráfico 6 com os artigos restantes após cada filtro aplicado.

Gráfico 6 - Artigos depois de todos os filtros



Fonte: O autor

3.6. Mapeamento dos dados

O último passo do processo de mapeamento sistemático da literatura envolve a categorização dos artigos relevantes em categorias baseadas em esquemas de

classificação bem definidos. Cada artigo relevante foi avaliado com a finalidade de categorizá-lo de acordo com cada esquema de classificação: técnica de teste de software estudada e algoritmo/técnica de inteligência artificial utilizada. Para isso, foi realizada uma leitura diagonal do conteúdo de cada artigo, a fim de identificar as informações necessárias para a categorização.

4. Resultados e discussão

Neste capítulo, descreveremos e discutiremos os resultados alcançados.

4.1. Técnicas de teste de software

Nesta seção, descreveremos as principais técnicas de teste de software que aparecem nos artigos selecionados.

Priorização de casos de teste (TCP): Segundo Elbaum *et al.* [14] A priorização de casos de teste envolve ordenar um conjunto de testes para maximizar algum objetivo específico, como antecipar a detecção de falhas ou a cobertura de código testável.

Seleção de casos de teste (TCS): É uma técnica utilizada para identificar casos de teste de acordo com um critério de seleção, o qual busca cobrir algum aspecto do software, assumindo que a cobertura levará à detecção de falhas. Um ponto importante é que ela depende das propriedades que devem ser testadas, dos critérios de cobertura e dos tipos de falhas que ele deseja detectar [15].

Predição de falhas: A predição de falhas é a técnica de prever o número de falhas em um código com base em suas características. Esse processo é uma tarefa complexa e requer uma avaliação cuidadosa de vários fatores, incluindo tamanho do código, complexidade ciclomática de McCabe, complexidade de Halstead entre outras [16].

Detecção de defeitos: A detecção de defeitos em software é um processo que visa prever áreas do código que possam conter defeitos. Isso ajuda os desenvolvedores a alocar seus esforços de teste verificando primeiro o código potencialmente problemático [17].

Localização de falhas: A localização de falhas consiste em identificar falhas em um programa para que possam ser reparadas posteriormente. Esse processo de encontrar uma falha pode ser um desafio, pois pode ser difícil distinguir um sintoma (erro em cascata) de uma causa (falha real). Além disso, várias falhas podem funcionar em conjunto, o que complica ainda mais o processo de localização [18].

Geração de casos de teste (TCG): A geração de casos de teste é o processo de construção de casos de testes para detectar comportamentos defeituosos em um sistema de software [19].

Geração de dados de teste (TDG): A geração de dados de teste é o processo de criar novos dados para serem utilizados como entrada para os casos de testes existentes, para assim verificar o comportamento do software que está sendo testado [20].

Teste de mutação: O teste de mutação é uma técnica utilizada para avaliar a qualidade de uma suíte de testes, injetando falhas artificiais no código. Se uma suíte de testes encontrar todos os erros que foram inseridos artificialmente (os mutantes) e não encontrar falhas no original, é provável que o programa em teste esteja livre de erros e que a suíte tenha qualidade (capacidade de detecção de erros), apesar de isso também depender da qualidade das falhas artificiais inseridas no código [21].

Reparação de programas: A reparação de programas é um processo pelo qual erros de programação em código-fonte, configurações, testes ou outros artefatos são corrigidos automaticamente. Esse processo visa encontrar e corrigir rapidamente bugs e vulnerabilidades à medida que o software evolui e é implantado [22].

Crowdtesting: Com o *crowdtesting* é possível encontrar bugs no código e gerar *feedback* sobre o produto em diversos cenários, com diferentes configurações de dispositivos, versões de sistemas operacionais ou condições de rede. Com o *crowdtesting*, uma empresa pode testar seus projetos em ambientes, dispositivos e condições do mundo real [214].

A Tabela 5 abaixo mostra uma categorização detalhada de todos os tipos de técnicas de testes de software identificados nos artigos analisados. Ela também mostra, para cada tipo de técnica de teste de software, quais artigos que os estudam. A Tabela 5 fornece uma visão clara e estruturada do teste de software e dos artigos que o estudou. Nesta seção, descrevemos acima apenas as técnicas que mais aparecem nos artigos. Uma descrição detalhada das demais técnicas pode ser encontrada na literatura [27, 44, 45, 54, 68, 77, 78, 85, 112, 127, 130].

Tabela 5 - Técnicas de testes e os artigos que as estudam

Técnica	**Estudos**
Priorização de casos de teste	[26], [29], [30], [36], [57], [75], [85], [88], [99], [113], [122], [132], [154], [179], [188], [196]
Seleção de casos de teste	[24], [38], [57], [82], [99], [124]
Predição de falhas	[23], [25], [28], [31], [33], [35], [43], [50], [51], [64], [66], [74], [76], [84], [92], [102], [104], [110], [111], [127], [135], [152], [165], [171], [174], [175], [177], [180], [187], [191], [196], [199], [201], [204], [208], [209], [211]
Detecção de defeitos	[42], [60], [70], [77], [91], [92], [101], [117], [148], [168], [170], [182], [190], [192], [210]
Localização de falhas	[37], [46], [69], [90], [95], [97], [98], [105], [106], [109], [115], [121], [131], [134], [140], [142], [143], [144], [148], [150], [155], [161], [163], [166], [169], [181], [187], [192], [193], [202]
Geração de casos de teste	[39], [40], [41], [47], [59], [64], [83], [86], [93], [96], [103], [116], [120], [126], [129], [133], [139], [158], [167], [168], [172], [189], [194], [200], [212], [213]
Geração de dados de teste	[32], [41], [56], [58], [61], [67], [107], [114], [128], [159], [160], [198], [207]
Teste de mutação	[39], [56], [61], [63], [87], [100], [108], [123], [161], [162], [197]
Reparação de programas	[48], [89], [98], [134], [146], [148], [151], [169], [170]
Geração de suíte de teste	[77], [178], [184]

Redução da suíte de teste	[27], [34], [53], [55], [57], [138], [172], [173]
Predição de suíte de teste	[127]
Teste *flaky*	[78], [80], [94], [118]
Oráculo de teste	[54], [81], [121], [136], [141], [145], [153], [158], [185]
Teste *t-way*	[85], [103], [133]
Crowdtesting	[49], [52], [62], [65], [71], [72], [79], [96], [137], [140], [147], [155], [156], [157], [164], [186], [195], [203], [206]
Teste de vulnerabilidade	[68], [73], [125], [149], [183], [205]
Code smell	[112], [119], [176]
Bug Severity	[130]
Teste de desempenho	[45], [54]
Teste de experiência do usuário	[44]

Fonte: O autor

4.2. Técnicas/Algoritmos de inteligência artificial

Nesta seção, descreveremos as principais técnicas/algoritmos de inteligência artificial utilizados nos artigos selecionados.

Algoritmos Genéticos: Algoritmos Genéticos (GA) são algoritmos de otimização com e sem restrições baseados em um processo de seleção natural que imita a evolução biológica. Desenvolvido por John Holland e seus colaboradores nas décadas de 1960 e 1970, o GA é baseado na teoria da seleção natural de Charles Darwin [215].

Processamento de Linguagem Natural (NLP): É um ramo da Inteligência Artificial focado em como os computadores podem processar a linguagem como os humanos fazem. O NLP pode ser usado para tarefas simples de análise de texto, como classificação de documentos e análise de sentimento em blocos de texto, bem como tarefas mais avançadas, como resumir relatórios [216].

Rede Neural Convolucional (CNN): Uma Rede Neural Convolucional (CNN) é uma abordagem de aprendizado profundo amplamente utilizada para resolver problemas complexos. Elas são capazes de aprender automaticamente hierarquias de características a partir dos dados de entrada, o que as torna muito eficazes para tarefas como reconhecimento de imagens e análise de linguagem natural [218].

Random Forest: *Random Forest* é um algoritmo que treina várias árvores de decisão em paralelo e pode ser usado para tarefas de classificação e regressão. Isso o torna uma ferramenta poderosa para uma ampla variedade de aplicações [219].

A Tabela 6 abaixo apresenta uma categorização detalhada das técnicas e algoritmos de IA identificados nos artigos analisados. Ela também mostra, para cada técnica ou algoritmo de IA, os artigos que as utilizam. A Tabela 6 fornece uma visão clara e estruturada das técnicas e algoritmos de IA e dos artigos que as utilizam. Nesta seção, descrevemos acima apenas as técnicas que mais aparecem nos artigos. Uma descrição detalhada das demais técnicas pode ser encontrada na literatura [23, 26, 37, 42, 43, 45, 47, 48, 51, 56, 64, 70, 73, 77, 79, 85, 102, 103, 105, 114, 116, 126, 128, 133, 147, 174, 178, 179, 180, 182, 188, 189, 190, 191, 197, 200, 210, 212].

Tabela 6 - Técnicas / algoritmos de IA e os artigos que os utilizam

Técnica / Algoritmo	Estudos
Rede Neural Recorrente (RNN)	[37], [46], [86], [105], [106], [109], [123], [131], [137], [141], [153], [161], [175], [193], [205], [206]
Rede Neural Convolucional (CNN)	[31], [46], [48], [60], [74], [95], [96], [109], [120], [121], [142], [146], [147], [150], [153], [157], [159], [163], [164], [177], [193], [194], [204]
Memória de longo prazo de curto prazo (LSTM)	[70], [96], [97], [98], [111], [140], [141], [142], [150], [152], [161], [169],[175], [186], [194], [204], [205]
Random forest	[35], [43], [44], [49], [50], [63], [76], [80], [84], [94], [96], [99], [108], [110], [112], [118], [119], [127], [166], [183], [201], [206]
Rede Grafos Neurais (GNN)	[51], [90], [125], [169]

Aprendizado por Reforço Profundo (DRL)	[77], [114]
Rede neural profunda (DNN)	[42], [46], [74], [75], [76], [95], [104], [144], [160], [165], [180], [199]
Algoritmo genético (GA)	[29], [30], [32], [36], [38], [39], [41], [55], [58], [61], [67], [79], [83], [87], [88], [93], [100], [101], [107], [113], [115], [122], [124], [132], [138], [139], [145], [147], [149], [181], [184], [192], [197], [198], [200], [207], [213]
Máquina de vetores de suporte (SVM)	[26], [27], [33], [43], [50], [63], [68], [76], [92], [94], [96], [101], [112], [117], [118], [119], [127], [135], [180], [187], [199], [201], [203], [206], [208], [209], [211]
Algoritmo Genético Híbrido (HGA)	[40], [56]
Algoritmo guloso	[79], [196]
Algoritmo de Otimização de Baleias (WOA)	[126]
Fuzzy c-means	[56]
Processamento de linguagem natural (NLP)	[24], [62], [65], [71], [72], [78], [79], [81], [89], [96], [97], [120], [129], [130], [137], [151], [155], [156], [158], [170], [185], [194], [195], [205]
Rede de Grafos Convolutivos (GCN)	[52], [66], [102], [143], [176]
K-means	[53], [55], [137], [157], [164], [172], [197], [203]
Algoritmo Genético Guiado (GGA)	[47], [59], [184]
Rede Adversária Generativa (GAN)	[105], [148], [152]
X-means	[48]
Redes Neurais	[54], [91], [111], [135], [136], [140], [145], [154], [174], [195], [196], [201], [203], [208]
KNN	[25], [27], [28], [43], [50], [63], [68], [94], [102], [112], [118], [180], [183], [203], [211]
Árvore de decisão	[27], [28], [43], [50], [63], [92], [94], [110], [112],

	[118], [119], [127], [171], [180], [183], [187], [195], [201], [208], [211]
Regressão logística	[35], [43], [63], [68], [76], [80], [102], [110], [118], [119], [127], [201]
Regressão Polinomial	[128]
Regressão Linear	[34], [180], [195]
Multilayer perceptron	[43], [96], [109], [110], [112], [118], [127], [193], [199], [211]
Q-learning	[45], [97], [120], [167]
Rede bayesiana	[35], [43], [50], [57], [63], [76], [94], [96], [110], [118], [127], [180], [187], [195], [201], [206], [208], [211]
Simulated annealing (SA)	[115], [174], [202]
Algoritmo JAYA	[133]
Agrupamento hierárquico (AHC)	[116], [168]
Algoritmo de Agrupamento Aglomerativo (ACA)	[197]
Redes Convolucionais Temporais (TCN)	[114]
Algoritmo de polinização de flores (FPA)	[103]
Otimização por Colônia de Formigas (ACO)	[200]
XGBoost	[64], [69], [80], [162]
K-Medoids	[147]
Rede de Funções de Base Radial	[128]
Redes de Isomorfismo em grafos (GIN)	[73]
Cuckoo search	[29], [179]
Dragonfly Algorithm	[85]
Bagging	[43], [68], [199]

Boosting	[43], [50], [68], [82], [92], [99], [199], [201], [208]
Análise de Componentes de Transferência (TCA)	[182]
Algoritmo de Otimização por Enxame (PSO)	[173], [174], [209]
Isolation Forest	[23]
Random Approximate Reduct (RAR)	[191]
Prioritized Sweeping (PS)	[210]
Temporal Difference (TD)	[210]
Differential Evolution (DE)	[189], [213]
Colônia de Abelhas Artificial (ABC)	[178]
Algoritmo de Reprodução Assexuada (ARA)	[179]
Algoritmo de Cardume de Peixes Artificial (AFSA)	[188]
Algoritmo de Evolução de Previsão Cinza (GPE)	[212]
Planejamento de Monte Carlo Parcialmente Observável (POMCP)	[190]

Fonte: O autor

5. Mapeamento do campo de estudo

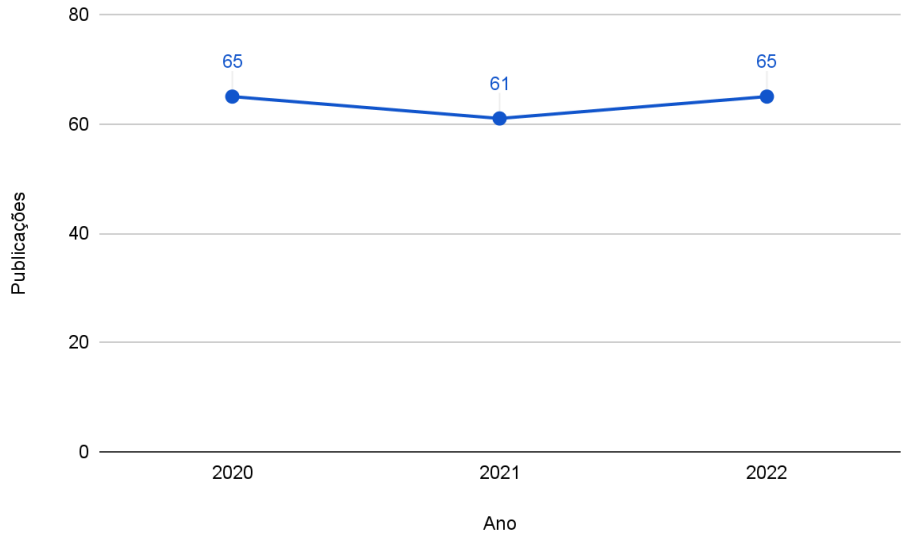
Neste estudo apresentamos um mapeamento sistemático das técnicas de teste de software utilizando algoritmos de inteligência artificial. Para esse estudo, foram coletados 198 artigos relevantes, de acordo com os esquemas de classificação definidos anteriormente, fornecendo uma visão abrangente do campo de pesquisa, destacando as principais tendências de publicação e identificando possíveis lacunas de pesquisa. Essas informações podem ser valiosas para orientar futuros estudos e avançar no estado da arte das técnicas de teste de software

utilizando inteligência artificial, fornecendo *insights* importantes para a comunidade acadêmica e profissional na área de engenharia de software.

5.1. Tendências de publicação

No Gráfico 7, é ilustrado o nível de atividade no campo de estudo ao longo dos últimos 3 anos, respondendo à primeira pergunta de pesquisa, "Qual é o número anual de publicações nessa área de pesquisa?". Nele, podemos ver que houve uma aparente estagnação nesse período, sem grandes oscilações na quantidade de publicações. Isso pode indicar a necessidade de um maior impulso na pesquisa e exploração desse tema, buscando novas abordagens para avançar no estado da arte das técnicas de teste de software utilizando inteligência artificial. A continuidade do interesse nesse campo e a investigação de possíveis lacunas de pesquisa podem ser oportunidades promissoras para futuras pesquisas e avanços na área.

Gráfico 7 - Artigos publicados ano a ano



Fonte: O autor

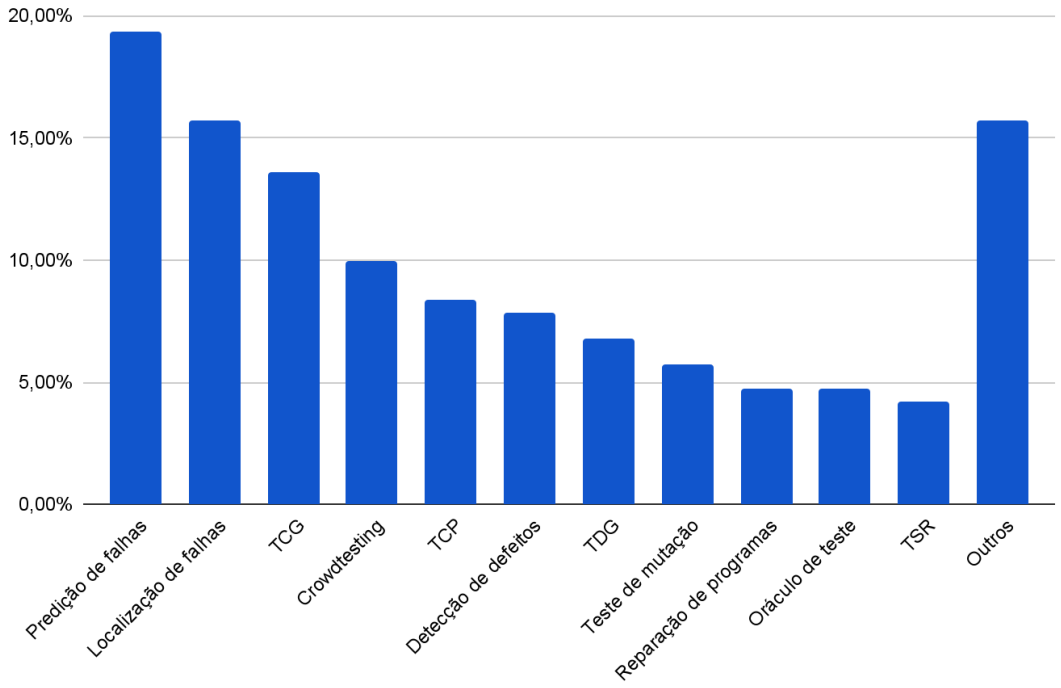
5.2. Técnicas de teste de software estudadas

Para responder à terceira pergunta de pesquisa, "Quais são as principais técnicas de testes de software estudadas?", realizamos uma leitura diagonal em

todos os artigos selecionados, para identificar as principais técnicas de testes de software estudadas nos artigos selecionados. Com isso, identificamos que as técnicas mais utilizadas são: Predição de Falhas, presente em 19,37% dos artigos analisados, Localização de Falhas, presente em 15,71%, Geração de Casos de Teste, presente em 13,61%, *Crowdtesting*, presente em 9,95%, Priorização de Casos de Teste, presente em 8,38% e Detecção de Defeitos, presente em 7,85%. No Gráfico 9, podemos observar com mais detalhes os dados referentes a cada técnica identificada nos artigos analisados no mapeamento sistemático.

É importante ressaltar que a soma das porcentagens de todas as técnicas de teste de software mencionadas ultrapassa os 100%. Isso acontece porque alguns artigos estudam mais de uma técnica em suas pesquisas.

Gráfico 9 - Artigos publicados ano a ano



Fonte: O autor

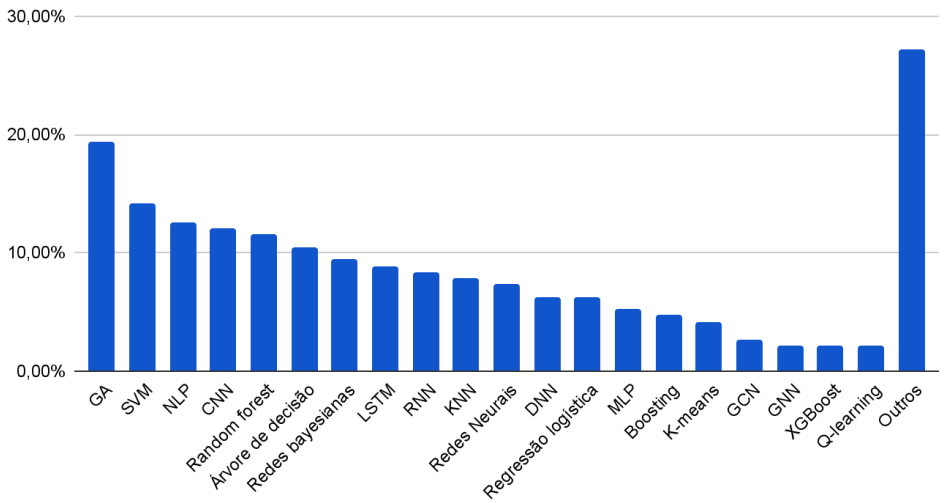
5.3. Algoritmos / Técnicas de inteligência artificial utilizadas

Para responder à segunda pergunta de pesquisa, "Quais são as principais técnicas/algoritmos de inteligência artificial utilizadas?", realizamos uma leitura

diagonal em todos os artigos selecionados. Identificamos que as técnicas mais utilizadas são: Algoritmo Genético (GA), presente em 19,37% dos artigos, Máquina de Vetores de Suporte (SVM), presente em 14,14%, Processamento de Linguagem Natural (NLP), presente em 12,57%, Redes Neurais Convolucionais (CNN), presentes em 12,04%, Random Forest, presente em 11,52% e Árvore de Decisão, presente em 10.47%. No Gráfico 8, podemos observar com mais detalhes os dados referentes a cada algoritmo ou técnica identificada nos artigos analisados no mapeamento sistemático.

É importante destacar que a soma da porcentagem de todas as técnicas e algoritmos mencionados ultrapassa os 100%. Isso ocorre porque alguns artigos utilizam mais de uma técnica ou algoritmo em seus estudos. Isso significa que, em vez de se limitar a apenas uma abordagem, esses artigos combinam diferentes técnicas e algoritmos para obter resultados mais precisos e abrangentes.

Gráfico 8 - Artigos publicados ano a ano



Fonte: O autor

6. Trabalhos relacionados

Lima *et al.* [12] examinaram a literatura disponível em Teste de Software aplicando algoritmos de Inteligência Artificial no período de 2017 a 2020. Esses artigos examinados foram obtidos a partir dos bancos de dados da Scopus, Elsevier,

Web of Science e Google Scholar. Após a coleta das informações, elas foram classificadas com base na abordagem de IA utilizada e o tipo de teste ao qual são aplicadas. Neste trabalho, Lima *et al.* [12] concluíram que os algoritmos de IA são usados, na maioria dos casos, para ajudar a cobrir a maior parte do código com a menor quantidade de casos de testes. Os 3 casos principais foram o de encontrar maneiras eficientes de cobrir o código, priorizar casos de teste e reduzir a quantidade de casos de teste.

Mayeda e Andrews [217] examinaram a literatura disponível em avaliação de Teste de Software no período de 2007 a 2017, obtendo os artigos a partir das bases de dados ACM, IEEE Xplore, Springer e Wiley, o qual resultou em 335 artigos após a realização dos filtros de ano da publicação, título e remoção de artigos duplicados. Após a coleta das informações, esses artigos foram organizados com base no método de avaliação e se usa mutação em sua análise. Ao final do trabalho, Mayeda e Andrews concluíram que há uma necessidade de pesquisa focada sobre como as técnicas de teste devem ser avaliadas, pois a maioria das avaliações feitas eram de baixa qualidade.

Lima *et al*. realizaram uma revisão da literatura de forma superficial sobre testes de software utilizando inteligência artificial, porém não foi aplicado um modelo de revisão sistemática. Por outro lado, Mayeda e Andrews conduziram um mapeamento sistemático da literatura, assim como proposto por Petersen *et al.* [13], mas com foco na avaliação das técnicas de testes de software. Neste estudo, realizamos um mapeamento sistemático da literatura, seguindo o modelo proposto por Petersen *et al*. [13], com o objetivo de investigar as técnicas de testes de software utilizando inteligência artificial em estudos publicados entre 2020 e 2022.

7. Conclusão

Este artigo apresenta um mapeamento sistemático da literatura de técnicas de teste de software utilizando IA, fornecendo uma visão geral do estado atual da arte e identificando lacunas de pesquisa que precisam ser exploradas.

O trabalho também fornece aos pesquisadores informações sobre quais técnicas de teste de software que utilizam IA são mais exploradas e onde obter trabalhos relacionados a essas técnicas com base no estado da arte. Isso torna mais fácil para os pesquisadores identificar publicações relevantes para seu campo de estudo.

Neste estudo, também verificamos que as técnicas de teste de software que utilizam algoritmos de IA mais exploradas são Predição de Falhas (19,37% dos artigos analisados), Localização de Falhas (15,71%), Geração de Casos de Teste (13,61%), *Crowdtesting* (9,95%) e Priorização de Casos de Teste (8,38%). Além disso, os algoritmos de IA mais utilizados nesses estudos são Algoritmo Genético (GA) (19,37% dos artigos), Máquina de Vetores de Suporte (SVM) (14,14%), Processamento de Linguagem Natural (NLP) (12,57%), Redes Neurais Convolucionais (CNN) (12,04%) e Random Forest (11,52%).

Como resultado, nosso trabalho contribui para o avanço da pesquisa em técnicas de teste de software utilizando IA, oferecendo uma base sólida para futuras investigações. Identificando lacunas de pesquisa e fornecendo informações sobre as técnicas de teste de software mais estudadas, além de ajudar os pesquisadores a direcionar seus esforços para áreas que ainda precisam ser mais exploradas.

8. Trabalhos futuros

Este trabalho alcançou resultados significativos na identificação do estado da arte de técnicas de teste de software utilizando IA. Com base nos resultados obtidos, surgiram algumas possibilidades para trabalhos futuros.

Algumas dessas possibilidades incluem a manutenção do mapeamento sistemático da literatura para os anos seguintes a 2022, a ampliação do escopo desta pesquisa para incorporar outras bases de dados e estudos pagos e a realização de revisões sistemáticas focadas em técnicas específicas para detalhar os resultados obtidos com essas revisões. Essas possibilidades podem ajudar a

expandir ainda mais o conhecimento sobre técnicas de teste de software utilizando IA.

9. Referências Bibliográficas

[1] KITCHENHAM, Barbara et al. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. Durham, Reino Unido: EBSE Technical Report, 2007.

[2] JAMIL, M. A. et al. **Software Testing Techniques: A Literature Review**. Mecca, Arábia Saudita: 6th International Conference on Information and Communication Technology for The Muslim World, 2016.

[3] SHAHROKNI, Ali; FELDT, Robert. **A Systematic Review of Software Robustness**. Information and Software Technology, vol. 55, no 1, 2013

[4] QAZI, Asad Masood; et al. **A systematic review of use cases based software testing techniques**. International Journal of Software Engineering and Its Applications, v. 10, 2016

[5] SOUZA, Simone R. S.; et al. **Research in concurrent software testing: a systematic review**. In: Proceedings of the Workshop on Parallel and Distributed Systems Testing, Analysis, and Debugging - PADTAD ’11. ACM Press, 2011

[6] **SOFTWARE testing**. Disponível em: https://en.wikipedia.org/wiki/Software_testing. Acesso em: 29 mar. 2023.

[7] **WHAT is software testing?** Disponível em: https://www.ibm.com/topics/software-testing. Acesso em: 29 mar. 2023.

[8] PRYTULENETS, Alesya. **A Brief History of Software Testing**. DogQ Blog, 9 fev. 2022. Disponível em: https://dogq.io/blog/a-brief-history-of-software-testing/. Acesso em: 29 mar. 2023.

[9] MEERTS, Joris; GRAHAM, Dorothy. **The History of Software Testing**. Testing References. Disponível em: https://www.testingreferences.com/testinghistory.php. Acesso em: 29 mar. 2023.

[10] ANJILA P K, Fathima. Artificial Intelligence. In: KARTHIKEYAN, J.; SU HIE, Ting; YU JIN, Ng. **Learning Outcomes of Classroom Research.** 1. ed. New Delhi: L Ordine Nuovo Publication, 2021. p. 65-73.

[11] SMITH, Chris; McGUIRE, Brian; HUANG, Ting; YANG, Gary. **The History of Artificial Intelligence**. In: HISTORY OF COMPUTING. Washington, 2006. CSEP 590A.

[12] LIMA, Rui; CRUZ, António Miguel Rosado da; RIBEIRO, Jorge. **Artificial Intelligence Applied to Software Testing: A Literature Review.** In: IBERIAN CONFERENCE ON INFORMATION SYSTEMS AND TECHNOLOGIES (CISTI), 15, 2020, Seville.

[13] PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. **Systematic mapping studies in software engineering**. In: 12th International Conference on Evaluation and Assessment in Software Engineering. Anais… BCS Learning & Development Ltd., 2008. p. 68-77

[14] ELBAUM, S.; MALISHEVSKY, A. G.; ROTHERMEL, G. **Test Case Prioritization: A Family of Empirical Studies**. IEEE Transactions on Software Engineering, Lincoln, NE, v. 28, n. 2, p. 159-182, fev. 2002.

[15] GRINDAL, M.; LINDSTRÖM, B.; OFFUTT, J.; ANDLER, S. F. **An evaluation of combination strategies for test case selection**. Empir Software Eng, p. 583-611, 20 out. 2006. Editor: Per Runeson.

[16] SHERER, Susan A.. **Software fault prediction**. Journal Of Systems And Software, v. 29, n. 2, p. 97-105, maio 1995. Elsevier BV.

[17] LI, Jian; HE, Pinjia; ZHU, Jieming; LYU, Michael R.. **Software Defect Prediction via Convolutional Neural Network**. 2017 Ieee International Conference On Software Quality, Reliability And Security (Qrs), p. 318-328, jul. 2017. IEEE.

[18] THALLER, H. et al. **Towards Fault Localization via Probabilistic Software Modeling**. In: IEEE WORKSHOP ON VALIDATION, ANALYSIS AND EVOLUTION OF SOFTWARE TESTS (VST), London: IEEE, 2020. p. 24-27.

[19] ALAGARSAMY, S. et al. **A3Test: Assertion-Augmented Automated Test Case Generation**. Monash, 2023.

[20] MOHAN, Madhav. **Approaches for Test Data Generation in Software Testing**. GeeksforGeeks, 2021. Disponível em: https://www.geeksforgeeks.org/approaches-for-test-data-generation-in-software-testing/. Acesso em: 11 abr. 2023.

[21] REALES, P.; POLO, M.; FERNÁNDEZ-ALEMÁN, J. L.; TOVAL, A.; PIATTINI, M. **Mutation Testing**. IEEE Software, v. 31, n. 3, p. 30-35, 2014.

[22] LE GOUES, C.; PRADEL, M.; ROYCHOUDHURY, A.; CHANDRA, S. **Automatic Program Repair**. IEEE Software, v. 38, n. 4, p. 22-27, 2021.

[23] DING, Z.; XING, L. **Improved software defect prediction using Pruned Histogram-based isolation forest**. Reliability Engineering & System Safety, Dartmouth, v. 204, 2020.

[24] SUTAR, S.; KUMAR, R.; PAI, S.; BR, S. **Regression Test cases selection using Natural Language Processing**. In: 2020 INTERNATIONAL CONFERENCE ON INTELLIGENT ENGINEERING AND MANAGEMENT (ICIEM), London, 2020. p. 301-305.

[25] ALI, A. R.; REHMAN, A. U.; NAWAZ, A.; ALI, T. M.; ABBAS, M. **An Ensemble Model for Software Defect Prediction**. In: 2022 2ND INTERNATIONAL CONFERENCE ON DIGITAL FUTURES AND TRANSFORMATIVE TECHNOLOGIES (ICODT2), Rawalpindi, 2022. p. 1-5.

[26] WEI, C.; SUN, Y.; SHENG, Y.; JIANG, S. **Machine Learning based Combinatorial Test Cases Ordering Approach**. In: 2021 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND ARTIFICIAL INTELLIGENCE (SEAI), 2021, Xiamen, 2021. p. 37-42.

[27] SAPUTRA, M.C.; KATAYAMA,T. **Code Coverage Similarity Measurement Using Machine Learning for Test Cases Minimization**. In: 2020 IEEE 9TH GLOBAL CONFERENCE ON CONSUMER ELECTRONICS (GCCE), Kobe, 2020. p. 287-291.

[28] AMJAD, H.M.W; RANA, Z.A **Using Developer Factors and Horizontal Partitioning to Recommend Bug Severity in Open-Source Software Projects**. In:2022 17TH INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES(ICET), Swabi, 2022. p. 130-135.

[29] GUPTA, S; CHOPRA, S; ARORA, M. **Implementation of Efficient Test Case Optimization Technique Using Meta-Heuristic Algorithm**. In:2021 9TH INTERNATIONAL CONFERENCE ON RELIABILITY INFOCOM TECHNOLOGIES AND OPTIMIZATION(TRENDS AND FUTURE DIRECTIONS)(ICRITO), Noida, 2021. p. 1-4.

[30] SWATI, B. **Automated Test Case Prioritization and Evaluation using Genetic Algorithm**. In: 2022 INTERNATIONAL CONFERENCE ON COMPUTING, COMMUNICATION, SECURITY AND INTELLIGENT SYSTEMS (IC3SIS), Kochi, 2022. p. 1-5.

[31] ZHENG, W.; TAN, L.; LIU, C. **Software Defect Prediction Method Based on Transformer Model**. In: 2021 IEEE INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND COMPUTER APPLICATIONS (ICAICA), Dalian, 2021. p. 670-674.

[32] NEETU, G. **ACTUM–tool for automatic class testing using meta-heuristics**. In: 2022 10TH INTERNATIONAL CONFERENCE ON RELIABILITY, INFOCOM TECHNOLOGIES AND OPTIMIZATION (TRENDS AND FUTURE DIRECTIONS) (ICRITO), Noida, 2022. p. 1-4.

[33] HUSIN, T.F.; PRIBADI, M.R.; YOHANNES. **Implementation of LSSVM in Classification of Software Defect Prediction Data with Feature Selection**. In:

2022 9TH INTERNATIONAL CONFERENCE ON ELECTRICAL ENGINEERING, COMPUTER SCIENCE AND INFORMATICS (EECSI), Jakarta, 2022. p. 126-131.

[34] TANEJA, D.; SINGH, R.; SINGH, A.; MALIK, H. **A Novel technique for test case minimization in object oriented testing**. Procedia Computer Science, Sonipat, v. 167, p. 2221-2228, 2020.

[35] KOMALASARI, A; CANDRA, M.Z.C. **Improving Defect Prediction Using Combination of Software Metrics**. In: 2022 INTERNATIONAL CONFERENCE ON DATA AND SOFTWARE ENGINEERING(ICoDSE), Denpasar, 2022. p. 89-94.

[36] ROSENBAUER,L; STEIN,A; PÄTZEL,D; HÄHNER,J. **XCSF with Experience Replay for Automatic Test Case Prioritization**. In: 2020 IEEE SYMPOSIUM SERIES ON COMPUTATIONAL INTELLIGENCE(SSCI), Canberra, 2020. p. 1307-1314.

[37] ZHANG, Y.; ZHOU, J.; HU, J. **Research on Defect Location Method of C Language Code Based on Deep Learning**. In: 2021 16TH INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE & EDUCATION (ICCSE), Lancaster, 2021. p. 360-365.

[38] BENITO-PAREJO, M.; MERAYO, M.G. **Using Genetic Algorithms To Select Test Cases For Finite State Machines With Timeouts**. In: 2021 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC), Kraków, 2021. p. 2403-2410.

[39] RAMÍREZ,A; DELGADO-PÉREZ,P; VALLE-GÓMEZ,K.J; MEDINA-BULO,I; ROMERO,J.R **Interactivity in the Generation of Test Cases with Evolutionary Computation**. In: 2021 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION(CEC), Kraków, 2021. p. 2395-2402.

[40] S,P; R,S. **Automatic Test Case Generation Using Hybrid Genetic Algorithm**. In: 2021 SECOND INTERNATIONAL CONFERENCE ON

ELECTRONICS AND SUSTAINABLE COMMUNICATION SYSTEMS(ICESC), Coimbatore, 2021. p. 1549-1556.

[41] ANH,B.T.M **Enhanced Genetic Algorithm for Automatic Generation of Unit and Integration Test Suite**. In: 2020 RIVF INTERNATIONAL CONFERENCE ON COMPUTING AND COMMUNICATION TECHNOLOGIES(RIVF), Ho Chi Minh City, 2020. p. 1-6.

[42] QU,T; LIU,W; ZHENG,W; TAO,W **Software Defect Detection Method Based on Graph Structure and Deep Neural Network**. In: 2022 INTERNATIONAL CONFERENCE ON ASIAN LANGUAGE PROCESSING(IALP), Singapore, 2022. p. 395-400.

[43] JACOB,R.J; KAMAT,R.J; SAHITHYA,N.M; JOHN,S.S; SHANKAR,S.P **Voting Based Ensemble Classification for Software Defect Prediction**. In: 2021 IEEE MYSORE SUB SECTION INTERNATIONAL CONFERENCE(MysuruCon), Hassan, 2021. p. 358-365.

[44] BIRINGA, C.; KUL, G. **Automated User Experience Testing through Multi-Dimensional Performance Impact Analysis**. In: 2021 IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATION OF SOFTWARE TEST (AST), 2021, Madrid, 2021. p. 125-128.

[45] MOGHADAM, M.H.; SAADATMAND, M.; BORG, M.; BOHLIN, M.; LISPER, B. Poster: **Performance Testing Driven by Reinforcement Learning**. In: 2020 IEEE 13TH INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VALIDATION AND VERIFICATION (ICST), Porto, 2020. p. 402-405.

[46] DUTTA,A. **Poster:EBFL-An Ensemble Classifier based Fault Localization**. In: 2022 IEEE CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION(ICST), Valencia, 2022. p. 473-476.

[47] SOLTANI,M; PANICHELLA,A; VAN DEURSEN,A. **Search-Based Crash Reproduction and Its Impact on Debugging**. In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, v46, n12, p1294-1317, 2020.

[48] LIU,K; KIM,D; BISSYANDÉ,T.F; YOO,S; LE TRAON,Y. **Mining Fix Patterns for FindBugs Violations**. In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, v47, n1, p. 165-188, 2021.

[49] ALAZZAM,I; ALEROUD,A; AL LATIFAH,Z; KARABATIS,G. **Automatic Bug Triage in Software Systems Using Graph Neighborhood Relations for Feature Augmentation**. In: IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS, v7, n5, p. 1288-1303, 2020.

[50] KHAN,F; KANWAL,S; ALAMRI,S; MUMTAZ,B. **Hyper-Parameter Optimization of Classifiers Using an Artificial Immune Network and Its Application to Software Bug Prediction**. In: IEEE ACCESS, v8, p. 20954-20964 2020.

[51] XU,J; WANG,F; Al,J. **Defect Prediction With Semantics and Context Features of Codes Based on Graph Representation Learning**. In: IEEE TRANSACTIONS ON RELIABILITY, v70, n2, p. 613-625, 2021.

[52] ZAIDI, S.F.A.; WOO, H.; LEE, C.-G. **A Graph Convolution Network-Based Bug Triage System to Learn Heterogeneous Graph Representation of Bug Reports**. In: IEEE ACCESS, v. 10, p. 20677-20689, 2022.

[53] CHETOUANE,N; WOTAWA,F; FELBINGER,H; NICA,M. **On Using k-means Clustering for Test Suite Reduction**. In: 2020 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION WORKSHOPS(ICSTW), Porto, 2020. p. 380-385.

[54] PORRES,I; AHMAD,T; REXHA,H; LAFOND,S; TRUSCAN,D. **Automatic exploratory performance testing using a discriminator neural network**. In: 2020 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION WORKSHOPS(ICSTW), Porto, 2020. p. 105-113.

[55] XIA,C; ZHANG,Y; HUI,Z. **Test Suite Reduction via Evolutionary Clustering**. In: IEEE ACCESS, v9, p.28111-28121, 2021.

[56] DANG,X; GONG,D; YAO,X; TIAN,T; LIU,H **Enhancement of Mutation Testing via Fuzzy Clustering and Multi-Population Genetic Algorithm**. In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, v48, n6, p2141-2156, 2022.

[57] PAN,R; ZHANG,Z; LI,X; CHAKRABARTY,K; GU,X. **Black-Box Test-Cost Reduction Based on Bayesian Network Models**. In: IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, v40, n2, p. 386-399, 2021.

[58] YAO,X; GONG,D; LI,B; DANG,X; ZHANG,G. **Testing Method for Software With Randomness Using Genetic Algorithm**. In: IEEE ACCESS, v8, p.61999-62010, 2020.

[59] BERGEL,A; MUÑOZ,I.S. **Beacon: Automated Test Generation for Stack-Trace Reproduction using Genetic Algorithms**. In: 2021 IEEE/ACM 14TH INTERNATIONAL WORKSHOP ON SEARCH-BASED SOFTWARE TESTING(SBST), Madrid, 2021. p. 1-7.

[60] TEHRANIJAMSAZ, A.; KHALEEL, M.; AKBARI, R.; JANNESARI, A. **DeepRace: A learning-based data race detector**. In: 2021 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION WORKSHOPS (ICSTW), Porto de Galinhas, 2021. p. 226-233.

[61] DANG,X; YAO,X; GONG,D; TIAN,T; SUN,B **Multi-Task Optimization-Based Test Data Generation for Mutation Testing via Relevance of Mutant Branch and Input Variable**. In: IEEE ACCESS, v8, p. 144401-144412, 2020.

[62] LANDING,C; TAHVILI,S; HAGGREN,H; LANGKVIS,M; MUHAMMAD,A; LOUFI, A. **Cluster-Based Parallel Testing Using Semantic Analysis**. In: 2020 IEEE INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE TESTING(AITest), Oxford, 2020. p. 99-106.

[63] BRITO,C; DURELLI,V.H.S; DURELLI,R.S; SOUZA,S.R.S.d; VINCENZI,A.M.R; DELAMARO,M.E. **A Preliminary Investigation into Using**

Machine Learning Algorithms to Identify Minimal and Equivalent Mutants. In: 2020 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION WORKSHOPS(ICSTW), Porto, 2020. p. 304-313.

[64] HERSHKOVICH,E; STERN,R; ABREU,R; ELMISHALI,A. **Prioritized Test Generation Guided by Software Fault Prediction**. In: 2021 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION WORKSHOPS(ICSTW), Porto de Galinhas, 2021. p. 218-225.

[65] WANG,J; YANG,Y; WANG,S; CHEN,C; WANG,D; WANG,Q **Context-Aware Personalized Crowdtesting Task Recommendation**. In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, v48, n8, p.3131-3144, 2022.

[66] ŠIKIĆ,L; KURDIJA,A.S; VLADIMIR,K; ŠILIĆ,M **Graph Neural Network for Source Code Defect Prediction**. In: IEEE ACCESS, v10, p.10402-10415, 2022.

[67] FAN,S; YAO,N; WAN,L; MA,B; ZHANG,Y. **An Evolutionary Generation Method of Test Data for Multiple Paths Based on Coverage Balance**. In: IEEE ACCESS, v9, p. 86759-86772, 2021.

[68] ZHANG, X.; XIE, H.; YANG, H.; SHAO, H.; ZHU, M. A. **General Framework to Understand Vulnerabilities in Information Systems**. In: IEEE ACCESS, v. 8, p. 121858-121873, 2020.

[69] YANG,B; HE,Y; LIU,H; CHEN,Y; JIN,Z. **A Lightweight Fault Localization Approach based on XGBoost**. In: 2020 IEEE 20TH INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY RELIABILITY AND SECURITY(QRS), Macau, 2020. p.168-179.

[70] LI,X; NIU,W; ZHANG,X; ZHANG,R; YU,Z; LI,Z **Improving Performance of Log Anomaly Detection With Semantic and Time Features Based on BiLSTM-Attention**. In: 2021 2ND INTERNATIONAL CONFERENCE ON ELECTRONICS COMMUNICATIONS AND INFORMATION TECHNOLOGY(CECIT), Sanya, 2021. p. 661-666.

[71] BERNARD,E; BOTELLA,J; AMBERT,F; LEGEARD,B; UTTING,M. **Tool Support for Refactoring Manual Tests**. In: 2020 IEEE 13TH INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VALIDATION AND VERIFICATION(ICST), Porto, 2020. p. 332-342.

[72] CHEN,H; HUANG,S; LIU,Y; LUO,R; XIE,Y. **An Effective Crowdsourced Test Report Clustering Model Based on Sentence Embedding**. In: 2021 IEEE 21ST INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY RELIABILITY AND SECURITY(QRS), Hainan, 2021. p. 888-899.

[73] XIA,X; WANG,Y; YANG,Y. **Source Code Vulnerability Detection Based On SAR-GIN**. In: 2021 2ND INTERNATIONAL CONFERENCE ON ELECTRONICS COMMUNICATIONS AND INFORMATION TECHNOLOGY(CECIT), Sanya, 2021. p. 1144-1149.

[74] CHEN,J et al. **Software Visualization and Deep Transfer Learning for Effective Software Defect Prediction**. In: 2020 IEEE/ACM 42ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING(ICSE), Seoul, 2020. p. 578-589.

[75] ABDELKARIM,M; ELADAWI,R. **TCP-Net:Test Case Prioritization using End-to-End Deep Neural Networks**. In: 2022 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION WORKSHOPS(ICSTW), Valencia, 2022. p. 122-129.

[76] CHEN,J; XU,J; CAI,S; WANG,X; GU,Y; WANG,S. **An efficient dual ensemble software defect prediction method with neural network**. In: 2021 IEEE INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING WORKSHOPS(ISSREW), Wuhan, 2021. p. 91-98.

[77] ROMDHANA,A; CECCATO,M; MERLO,A; TONELLA,P. **IFRIT:Focused Testing through Deep Reinforcement Learning**. In: 2022 IEEE CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION(ICST), Valencia, 2022. p. 24-34.

[78] FATIMA,S; GHALEB,T.A; BRIAND,L. **Flakify: A Black-Box Language Model-Based Predictor for Flaky Tests**. In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2022.

[79] ZHU,P; LI,Y; LI,T; REN,H; SUN,X. **Advanced Crowdsourced Test Report Prioritization Based on Adaptive Strategy**. In: IEEE ACCESS, v10, p.53522-53532, 2022.

[80] MARTINS,R; ABREU,R; LOPES,M; NADKARNI,J. **Supervised Learning for Test Suit Selection in Continuous Integration**. In: 2021 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION WORKSHOPS(ICSTW), Porto de Galinhas, 2021. p. 239-246.

[81] DINELLA,E; RYAN,G; MYTKOWICZ,T; LAHIRI,S.K. **TOGA:A Neural Method for Test Oracle Generation**. In: 2022 IEEE/ACM 44TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING(ICSE), Pittsburgh, 2022. p.2130-2141.

[82] SENCHENKO,A; PATTERSON,J; SAMUEL,H; ISPIR,D. **SUPERNOVA:Automating Test Selection and Defect Prevention in AAA Video Games Using Risk Based Testing and Machine Learning**. In: 2022 IEEE CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION(ICST), Valencia, 2022. p. 345-354.

[83] WANG,R; ARTHO,C; KRISTENSEN,L.M; STOLZ,V. **Multi-objective Search for Model-based Testing**. In: 2020 IEEE 20TH INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY RELIABILITY AND SECURITY(QRS), Macau, 2020. p. 130-141.

[84] ZHANG,T; DU,Q; XU,J; LI,J; LI,X **Software Defect Prediction and Localization with Attention-Based Models and Ensemble Learning**. In: 2020 27TH ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE(APSEC), Singapore, 2020. p. 81-90.

[85] AHMED,M; NASSER,A.B; ZAMLI,K.Z **Construction of Prioritized T-Way Test Suite Using Bi-Objective Dragonfly Algorithm**. In:IEEE ACCESS, v10, p. 71683-71698, 2022.

[86] WATSON,C; TUFANO,M; MORAN,K; BAVOTA,G; POSHYVANYK,D. **On Learning Meaningful Assert Statements for Unit Test Cases**. In: 2020 IEEE/ACM 42ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING(ICSE), Seoul, 2020. p. 1398-1409.

[87] WANG,X; ZHANG,S; LIU,F; FENG,L; ZHAO,Z. **MQP:Mutants Quality Prediction for Cost-Effective Mutation Testing**. In: 2021 IEEE 21ST INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY RELIABILITY AND SECURITY COMPANION(QRS-C), Hainan, 2021. p. 45-50.

[88] JIN, K.; LANO, K. **OCL-based test case prioritisation using AgileUML**. In: PROCEEDINGS OF THE 25TH INTERNATIONAL CONFERENCE ON MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS. Montreal, 2022. p. 607-611.

[89] LAJKÓ, M.; CSUVIK, V.; VIDÁCS, L. **Towards JavaScript program repair with generative pre-trained transformer (GPT-2)**. In: PROCEEDINGS OF THE THIRD INTERNATIONAL WORKSHOP ON AUTOMATED PROGRAM REPAIR. Pittsburgh, 2022. p. 61-68.

[90] LI,Z; TANG,E; CHEN,X; WANG,L; LI,X **Graph Neural Network based Two-Phase Fault Localization Approach**. In: PROCEEDINGS OF THE 13TH ASIA-PACIFIC SYMPOSIUM ON INTERNETWARE. Hohhot, 2022. p. 85-95.

[91] SHARMA,A; MELNIKOV,V; HÜLLERMEIER,E; WEHRHEIM,H **Property-driven testing of black-box functions**. In: PROCEEDINGS OF THE IEEE/ACM 10TH INTERNATIONAL CONFERENCE ON FORMAL METHODS IN SOFTWARE ENGINEERING. Pittsburgh, 2022. p. 113-123.

[92] SUH,A **Adapting bug prediction models to predict reverted commits at Wayfair**. In: PROCEEDINGS OF THE 28TH ACM JOINT MEETING ON

EUROPEAN SOFTWARE ENGINEERING CONFERENCE AND SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING. Boston, 2020. p. 1251-1262.

[93] WU, D.; ZHU, J.; TANG, S. **Application in Computer Software Testing Based on Artificial Intelligence Technology**. In: PROCEEDINGS OF THE 2020 INTERNATIONAL CONFERENCE ON COMPUTERS, INFORMATION PROCESSING AND ADVANCED EDUCATION. Ottawa, 2020. p. 177-181.

[94] PINTO,G; MIRANDA,B; DISSANAYAKE,S; D'AMORIM,M; TREUDE,C; BERTOLINO,A **What is the Vocabulary of Flaky Tests?** In: PROCEEDINGS OF THE 17TH INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES. Seoul, 2020. p. 492-502.

[95] ZHANG,J; XIE,R; YE,W; ZHANG,Y; ZHANG,S **ExplJust-in-Time Defect Prediction for Android Apps via Imbalanced Deep Learning Moiting Code Knowledge Graph for Bug Localization via Bi-directional Attention**. In: PROCEEDINGS OF THE 28TH INTERNATIONAL CONFERENCE ON PROGRAM COMPREHENSION. Seoul, 2020. p. 219-229.

[96] LIU,H; SHEN,M; JIN,J; JIANG,Y **Automated classification of actions in bug reports of mobile apps**. In: PROCEEDINGS OF THE 29TH ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS. New York, 2020. p. 128-140.

[97] PAN,M; HUANG,A; WANG,G; ZHANG,T; LI,X **Reinforcement learning based curiosity-driven testing of Android applications**. In: PROCEEDINGS OF THE 29TH ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS. Nanjing, 2020. p. 153-164.

[98] MENG,X; WANG,X; ZHANG,H; SUN,H; LIU,X **Improving fault localization and program repair with deep semantic features and transferred knowledge**. In: PROCEEDINGS OF THE 44TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. Pittsburgh, 2022. p. 1169-1180.

[99] BERTOLINO,A; GUERRIERO,A; MIRANDA,B; PIETRANTUONO,R; RUSSO,S **Learning-to-rank vs ranking-to-learn:strategies for regression testing in continuous integration**. In: PROCEEDINGS OF THE ACM/IEEE 42ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. Seoul, 2020. p. 1-12.

[100] WANG,X; YU,T; ARCAINI,P; YUE,T; ALI,S **Mutation-based test generation for quantum programs with multi-objective search**. In: PROCEEDINGS OF THE GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE. Boston, 2022. p. 1345-1353.

[101] CYNTHIA,S.T; ROY,B; MONDAL,D **Feature Transformation for Improved Software Bug Detection Models**. In: 15TH INNOVATIONS IN SOFTWARE ENGINEERING CONFERENCE. Gandhinagar, 2022.1-10.

[102] RAJNISH,K; BHATTACHARJEE,V; CHANDRABANSHI,V **Applying Cognitive and Neural Network Approach over Control Flow Graph for Software Defect Prediction**. In: 2021 THIRTEENTH INTERNATIONAL CONFERENCE ON CONTEMPORARY COMPUTING. Noida, 2021. p. 13-17.

[103] NASSER, Abdullah B. et al. **T-way Test Suite Generation Based on Hybrid Flower Pollination Algorithm and Hill Climbing**. In: 2021 10th INTERNATIONAL CONFERENCE ON SOFTWARE AND COMPUTER APPLICATIONS, Kuala Lumpur, 2021. p. 244-250.

[104] ZHAO, Kunsong et al. **Just-in-time defect prediction for Android apps via imbalanced deep learning model**. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, New York, 2021. p. 1447-1454.

[105] PADURARU, Ciprian; STEFANESCU, Alin; GHIMIS, Bogdan. **Testing multi-tenant applications using fuzzing and reinforcement learning**. In: ACM SIGSOFT INTERNATIONAL WORKSHOP ON LANGUAGES AND TOOLS FOR NEXT-GENERATION TESTING, New York, 2020. p. 1-6.

[106] PRADEL, Michael et al. **Scaffle: bug localization on millions of files**. In: ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS, New York, 2020. p. 225-236.

[107] CHENG, Mengfei; DING, Rui; HUO, Tingting. **A hyper-heuristic algorithm for test data generation**. In: INTERNATIONAL CONFERENCE ON BIG DATA RESEARCH, Harbin, 2022. p. 26-31.

[108] KAUFMAN, Samuel J. et al. **Prioritizing mutants to guide mutation testing**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, Pittsburgh, 2022. p. 1743-1754.

[109] XIE, Huan et al. **A universal data augmentation approach for fault localization**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, Pittsburgh, 2022. p. 48-60.

[110] ABAEI, Golnoush et al. **Improving software fault prediction in imbalanced datasets using the under-sampling approach**. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND COMPUTER APPLICATIONS, Melaka, 2022. p. 41-47.

[111] CHOPRA, Rahul; ROY, Shreoshi; MALHOTRA, Ruchika. **Transductive Instance Transfer Learning for Cross-Language Defect Prediction**. In: ASIA PACIFIC INFORMATION TECHNOLOGY CONFERENCE, New York, 2022. p. 176-182.

[112] PATNAIK, Archana; PADHY, Neelamadhab. **Does Code Complexity Affect the Quality of Real-Time Projects? Detection of Code Smell on Software Projects using Machine Learning Algorithms**. In: INTERNATIONAL CONFERENCE ON DATA SCIENCE, MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE, Windhoek, 2021, p. 178-185.

[113] VESCAN, Andreea et al. **On the use of evolutionary algorithms for test case prioritization in regression testing considering requirements**

dependencies. In: ACM INTERNATIONAL WORKSHOP ON AI AND SOFTWARE TESTING/ANALYSIS, New York, 2021. p. 1-8.

[114] HUURMAN, Steyn; BAI, Xiaoying; HIRTZ, Thomas. **Generating API Test Data Using Deep Reinforcement Learning**. In: IEEE/ACM INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING WORKSHOPS, Seoul, 2020. p. 541-544.

[115] SILVA-JUNIOR, Deuslirio et al. **Data-flow-based evolutionary fault localization**. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, Brno, 2020. p. 1963-1970.

[116] OLSTHOORN, Mitchell. **More effective test case generation with multiple tribes of AI**. In: ACM/IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING: COMPANION PROCEEDINGS, Pittsburgh, 2022. p. 286-290.

[117] FIGUEIREDO, Lorena P. de et al. **An automatic approach to detect problems in Android builds through screenshot analysis**. In: ACM/SIGAPP SYMPOSIUM ON APPLIED COMPUTING, Brno, 2022. p. 926-932.

[118] CAMARA, Bruno et al. **On the use of test smells for prediction of flaky tests**. In: BRAZILIAN SYMPOSIUM ON SYSTEMATIC AND AUTOMATED SOFTWARE TESTING, Joinville, 2021. p. 46-54.

[119] MARTINS, Luana et al. **Smart prediction for refactorings in the software test code**. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, Joinville, 2021. p. 115-120.

[120] COLLINS, Eliane et al. **Deep Reinforcement Learning based Android Application GUI Testing**. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, Joinville, 2021. p. 186-194.

[121] CHEN, Ke et al. **GLIB: towards automated test oracle for graphically-rich applications**. In: ACM JOINT MEETING ON EUROPEAN

SOFTWARE ENGINEERING CONFERENCE AND SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING, Athens, 2021. p. 1093-1104.

[122] LIMA, Jackson A. Prado; VERGILIO, Silvia R. **Multi-Armed Bandit Test Case Prioritization in Continuous Integration Environments: A Trade-off Analysis**. In: BRAZILIAN SYMPOSIUM ON SYSTEMATIC AND AUTOMATED SOFTWARE TESTING, Natal, 2020. p. 21-30.

[123] TUFANO, Michele et al. **DeepMutation: a neural mutation tool**. In: ACM/IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING: COMPANION PROCEEDINGS, Seoul, 2020. p. 29-32.

[124] JIA, Huihui; ZHANG, Cheng; WU, Sijie. **Multi-objective software test case selection based on density analysis**. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING, Guilin, 2022. p. 140-147.

[125] CHENG, Xiao et al. **Path-sensitive code embedding via contrastive learning for software vulnerability detection**. In: ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS, New York, 2022. p. 519-531.

[126] WANG, Jing; ZHAO, Weidong. **Automatic Test Case Generation Method Based on Improved Whale Optimization Algorithm**. In: INTERNATIONAL CONFERENCE ON INTELLIGENT SYSTEMS, METAHEURISTICS & SWARM INTELLIGENCE, Victoria, 2021. p. 7-16.

[127] PAN, Cong; PRADEL, Michael. **Continuous test suite failure prediction**. In: ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS, New York, 2021. p. 553-565.

[128] GONG, Dunwei et al. **Test Data Generation for Path Coverage of MPI Programs Using SAEO**. ACM Trans. Softw. Eng. Methodol., New York, v. 30, n. 2, p. 1-37, abr. 2021.

[129] TAN, Shin Hwei; LI, Ziqiang. **Collaborative bug finding for Android apps**. In: ACM/IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, Seoul, 2020. p. 1335-1347.

[130] AROKIAM, Jude; BRADBURY, Jeremy S. **Automatically predicting bug severity early in the development process**. In: ACM/IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING: NEW IDEAS AND EMERGING RESULTS, Seoul, 2020. p. 17-20.

[131] XUE, Feng. **Automated mobile apps testing from visual perspective**. In: ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS, New York, 2020. p. 577-581.

[132] ROSENBAUER, Lukas et al. **XCS as a reinforcement learning approach to automatic test case prioritization**. In: GGenerating API Test Data Using Deep ReinforcemENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE COMPANION, Cancún, 2020. p. 1798-1806.

[133] NASSER, Abdullah B. et al. **Latin Hypercube Sampling Jaya Algorithm based Strategy for T-way Test Suite Generation**. In: Proceedings of the 2020 9th International Conference on Software and Computer Applications. Langkawi, 2020. p. 105-109.

[134] YUAN, Yuan; BANZHAF, Wolfgang. **Toward Better Evolutionary Program Repair: An Integrated Approach**. ACM Trans. Softw. Eng. Methodol., New York, v. 29, n. 1, p. 5-57, 2020.

[135] SADAF, Saadia; IQBAL, Danish; BUHNOVA, Barbora. **AI-Based Software Defect Prediction for Trustworthy Android Apps**. In: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering. Gothenburg, 2022. p. 393-398.

[136] TSIMPOURLAS, Foivos; RAJAN, Ajitha; ALLAMANIS, Miltiadis. **Supervised learning over test executions as a test oracle**. In: Proceedings of the

36th Annual ACM Symposium on Applied Computing. New York, 2021. p. 1521-1531.

[137] LI, Linyi et al. **Clustering test steps in natural language toward automating test automation**. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, 2020. p. 1285-1295.

[138] COVIELLO, Carmen et al. **GASSER: Genetic Algorithm for teSt Suite Reduction**. In: Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement. Bari, 2020. p. 36-41.

[139] MORENO, Iván Arcuschin. **Search-based test generation for Android apps**. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings. Seoul, 2020. p. 230-233.

[140] ZHANG, Xing et al. **DeFault: mutual information-based crash triage for massive crashes**. In: Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, 2022. p. 635-646.

[141] JABBARVAND, Reyhaneh; MEHRALIAN, Forough; MALEK, Sam. **Automated construction of energy test oracles for Android**. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, 2020. p. 927-938.

[142] LI, Zeyan et al. **Actionable and interpretable fault localization for recurring failures in online service systems**. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, 2022. p. 996-1008.

[143] LI, Yi; WANG, Shaohua; NGUYEN, Tien N. **Fault localization to detect co-change fixing locations**. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, 2022. p. 659-671.

[144] CAO, Junming et al. **BugPecker: locating faulty methods with deep learning on revision graphs**. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. New York, 2021. p. 1214-1218.

[145] MOLINA, Facundo. **Applying learning techniques to oracle synthesis**. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. New York, 2021. p. 1153-1157.

[146] LUTELLIER, Thibaud et al. **CoCoNuT: combining context-aware neural translation models using ensemble for program repair**. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, 2020. p. 101-114.

[147] DU, Mingzhe et al. **SemCluster: a semi-supervised clustering tool for crowdsourced test reports with deep image understanding**. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, 2022. p. 1756-1759.

[148] LIU, Zhe. **Woodpecker: identifying and fixing Android UI display issues**. In: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings. Pittsburgh, 2022. p. 334-336.

[149] DASS, Shuvalaxmi; NAMIN, Akbar Siami. **Vulnerability coverage for adequacy security testing.** In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. Brno, 2020. p. 540-543.

[150] ZHANG, Ziqian et al. **UniRLTest: universal platform-independent testing with reinforcement learning via image understanding**. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, 2022. p. 805-808.

[151] YUAN, Wei et al. **CIRCLE: continual repair across programming languages**. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, 2022. p. 678-690.

[152] XING, Y.; QIAN, X.; GUAN, Y.; YANG, B.; ZHANG, Y. **Cross-project defect prediction based on G-LSTM model**. Pattern Recognition Letters, Beijing, v. 160, p. 50-57, 2022.

[153] IBRAHIMZADA, Ali Reza et al. **Perfect is the enemy of test oracle**. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, 2022. p. 70-81.

[154] TIUTIN, Cristina-Maria; VESCAN, Andreea. **Test case prioritization based on neural networks classification.** In: Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis. New York, 2022. p. 9-16.

[155] ZHAO, Yu et al. **ReCDroid+: Automated End-to-End Crash Reproduction from Bug Reports for Android Apps**. ACM Trans. Softw. Eng. Methodol., New York, v. 31, n. 3, p. 36-68, 2022.

[156] PICUS, Oskar; SERBAN, Camelia. **Bugsby: a tool support for bug triage automation**. In: Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis. New York, 2022. p. 17-20.

[157] CAO, Zhenfei et al. **STIFA: crowdsourced mobile testing report selection based on text and image fusion analysis**. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. New York, 2021. p. 1331-1335.

[158] LIN, Jun-Wei; JABBARVAND, Reyhaneh; MALEK, Sam. **Test transfer across mobile apps through semantic mapping**. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering. Irvine, 2020. p. 42-53.

[159] YAZDANIBANAFSHEDARAGH, Faraz; MALEK, Sam. **Deep GUI: black-box GUI input generation with deep learning**. In: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering. Irvine, 2022. p. 905-916.

[160] LI, Yuanchun et al. **Humanoid: a deep learning-based approach to automated black-box Android app testing**. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering. Beijing, 2020. p. 1070-1073.

[161] JEON, Juyoung; HONG, Shin. **Improving mutation-based fault localization with plausible-code generating mutation operators**. In: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering. Pohang, 2022. p. 1205-1207.

[162] MA, Wei et al. **MuDelta: Delta-Oriented Mutation Testing at Commit Time**. In: Proceedings of the 43rd International Conference on Software Engineering. London, 2021. p. 897-909.

[163] LI, Yi; WANG, Shaohua; NGUYEN, Tien N. **Fault Localization with Code Coverage Representation Learning**. In: Proceedings of the 43rd International Conference on Software Engineering. Piscataway, 2021. p. 661-673.

[164] COOPER, Nathan et al. **It Takes Two to Tango: Combining Visual and Textual Information for Detecting Duplicate Video-Based Bug Reports**. In: Proceedings of the 43rd International Conference on Software Engineering. Williamsburg, 2021. p. 957-969.

[165] QIAO, L.; LI, X.; UMER, Q.; GUO, P. **Deep learning based software defect prediction**. Neurocomputing, Beijing, v. 385, p. 100-110, 2020.

[166] KÜÇÜK, Yiğit; HENDERSON, Tim A. D.; PODGURSKI, Andy. **Improving Fault Localization by Integrating Value and Predicate Based Causal Inference Techniques.** In: Proceedings of the 43rd International Conference on Software Engineering. Cleveland, 2021. p. 649-660.

[167] ZHENG, Yan et al. **Automatic Web Testing Using Curiosity-Driven Reinforcement Learning**. In: Proceedings of the 43rd International Conference on Software Engineering. Tianjin, 2021. p. 423-435.

[168] STALLENBERG, Dimitri; OLSTHOORN, Mitchell; PANICHELLA, Annibale. **Improving test case generation for REST APIs through hierarchical clustering**. In: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering. Delft, 2022. p. 117-128.

[169] XU, Xuezheng; WANG, Xudong; XUE, Jingling. **M3V: multi-modal multi-view context embedding for repair operator prediction.** In: Proceedings of the 20th IEEE/ACM International Symposium on Code Generation and Optimization. Sydney, 2022. p. 266-277.

[170] JIANG, Nan; LUTELLIER, Thibaud; TAN, Lin. **CURE: Code-Aware Neural Machine Translation for Automatic Program Repair**. In: Proceedings of the 43rd International Conference on Software Engineering. Lafayette, 2021. p. 1161-1173.

[171] CHEN, J.; HU, K.; YANG, Y.; LIU, Y.; XUAN, Q. **Collective transfer learning for defect prediction**. Neurocomputing, Hangzhou, v. 416, p. 103-116, 2020.

[172] SUBASHINI B.; SUNDARAVADIVAZHAGAN B.; ASHIK M. **Amalgamation and characterization for test Suite enhancement in software testing using data mining techniques**. Materials Today: Proceedings, Madurai, 2021.

[173] DENEKE, A.; GIZACHEW ASSEFA, B.; KUMAR MOHAPATRA, S. **Test suite minimization using particle swarm optimization**. Materials Today: Proceedings, Ababa, v. 60, p. 229-233, 2022.

[174] KASSAYMEH, S.; AL-LAHAM, M.; AL-BETAR, M.A.; ALWESHAH, M.; ABDULLAH, S.; MAKHADMEH, S.N. **Backpropagation Neural Network optimization and software defect estimation modelling using a hybrid Salp Swarm optimizer-based Simulated Annealing Algorithm.** Knowledge-Based Systems, Bangi Selango, v. 244, 2022.

[175] PANDEY, S.K.; TRIPATHI, A.K. **BCV-Predictor: A bug count vector predictor of a successive version of the software system**. Knowledge-Based Systems, Varanasi, v. 197, 2020.

[176] WU, B.; LIANG, B.; ZHANG, X. **Turn tree into graph: Automatic code review via simplified AST driven graph convolutional network.** Knowledge-Based Systems, Suzhou, v. 252, 2022.

[177] BALASUBRAMANIAM D.S.; GOLLAGI D.S. **Software defect prediction via optimal trained convolutional neural network**. Advances in Engineering Software, Thiruvananthapuram, v. 169, 2022.

[178] SAHIN O.; AKAY B. **A Discrete Dynamic Artificial Bee Colony with Hyper-Scout for RESTful web service API test suite generation.** Applied Soft Computing, Kayseri, v. 104, 2021.

[179] BAJAJ A.; SANGWAN O.P. **Discrete cuckoo search algorithms for test case prioritization**. Applied Soft Computing, Auburn, v. 110, 2021.

[180] MANCHALA P.; BISI M. **Diversity based imbalance learning approach for software fault prediction using machine learning models**. Applied Soft Computing, Warangal, v. 124, 2022.

[181] XIAOBO Y.; BIN L.; SHIHAI W. **A Test Restoration Method based on Genetic Algorithm for effective fault localization in multiple-fault programs.** Journal of Systems and Software, Beijing, v. 172, 2021.

[182] ZHANG Y.; JIN D.; XING Y.; GONG Y. **Automated defect identification via path analysis-based features with transfer learning.** Journal of Systems and Software, Beijing, v. 166, 2020.

[183] CHEN J.; KUDJO P.K.; MENSAH S.; BROWN S.A.; AKORFU G. **An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection**. Journal of Systems and Software, Zhenjiang, v. 167, 2020.

[184] IBIAS A. **Using mutual information to test from Finite State Machines: Test suite generation**. Journal of Systems and Software, Madrid, v. 192, 2022.

[185] BLASI, A.; GORLA, A.; ERNST, M.D.; PEZZÈ, M.; CARZANIGA, A. **MeMo: Automatically identifying metamorphic relations in Javadoc comments for test automation**. Journal of Systems and Software, v. 181, 2021.

[186] ZHOU C.; LI B.; SUN X. **Improving software bug-specific named entity recognition with deep neural network**. Journal of Systems and Software, Yangzhou, v. 165, 2020.

[187] MARIANI L.; PEZZÈ M.; RIGANELLI O.; XIN R. **Predicting failures in multi-tier distributed systems**. Journal of Systems and Software, Milan, v. 161, 2020.

[188] XING Y.; WANG X.; SHEN Q. **Test case prioritization based on Artificial Fish School Algorithm**. Computer Communications, Beijing, v. 180, p. 295-302, 2021.

[189] DAI X.; GONG W.; GU Q. **Automated test case generation based on differential evolution with node branch archive**. Computers & Industrial Engineering, Wuhan, v. 156, 2021.

[190] AKBARINASAJI S.; KAVAKLIOGLU C.; BAŞAR A.; NEAL A. **Partially observable Markov decision process to generate policies in software defect management.** Journal of Systems and Software, Kanata, v. 163, 2020.

[191] JIANG F.; YU X.; GONG D.; DU J. **A random approximate reduct-based ensemble learning approach and its application in software defect prediction.** Information Sciences, Qingdao, v. 609, p. 1147-1168, 2022.

[192] GHOSH D.; SINGH J. **Spectrum-based multi-fault localization using Chaotic Genetic Algorithm**. Information and Software Technology, Bhubaneswar, v. 133, 2021.

[193] ZHANG Z.; LEI Y.; MAO X.; YAN M.; XU L.; ZHANG X. **A study of effectiveness of deep learning in locating real faults**. Information and Software Technology, Guilin, v. 131, 2021.

[194] KHALIQ Z.; FAROOQ S.U.; KHAN D.A. **A deep learning-based automated framework for functional User Interface testing.** Information and Software Technology, Kashmir, v. 150, 2022.

[195] SOLEIMANI NEYSIANI B.; BABAMIR S.M.; ARITSUGI M. **Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems.** Information and Software Technology, Kashan, v. 126, 2020.

[196] MAHDIEH M.; MIRIAN-HOSSEINABADI S.H.; ETEMADI K.; NOSRATI A.; JALALI S. **Incorporating fault-proneness estimations into coverage-based test case prioritization methods**. Information and Software Technology, Tehran, v. 121, 2020.

[197] WEI C.; YAO X.; GONG D.; LIU H. **Spectral clustering based mutant reduction for mutation testing**. Information and Software Technology, Xuzhou, v. 132, 2021.

[198] GONG D.; PAN F.; TIAN T.; YANG S.; MENG F. **A feedback-directed method of evolutionary test data generation for parallel programs**. Information and Software Technology, Xuzhou, v. 124, 2020.

[199] YU X.; KEUNG J.; XIAO Y.; FENG S.; LI F.; DAI H. **Predicting the precise number of software defects: Are we there yet?** Information and Software Technology, Wuhan, v. 146, 2022.

[200] ARORA V.; SINGH M.; BHATIA R. **Orientation-based Ant colony algorithm for synthesizing the test scenarios in UML activity diagram**. Information and Software Technology, Patiala, v. 123, 2020.

[201] ZHAO Y.; WANG Y.; ZHANG Y.; ZHANG D.; GONG Y.; JIN D. **ST-TLF: Cross-version defect prediction framework based transfer learning.** Information and Software Technology, Beijing, v. 149, 2022.

[202] ALMHANA R.; KESSENTINI M.; MKAOUER W. **Method-level bug localization using hybrid multi-objective search**. Information and Software Technology, v. 131, 2021.

[203] TONG Y.; ZHANG X. **Crowdsourced test report prioritization considering bug severity**. Information and Software Technology, Suzhou, v. 139, 2021.

[204] ZOU Q.; LU L.; YANG Z.; GU X.; QIU S. **Joint feature representation learning and progressive distribution matching for cross-project defect prediction.** Information and Software Technology, Guangzhou, v. 137, 2021.

[205] JEON S.; KIM H.K. **AutoVAS: An automated vulnerability analysis system with a deep learning approach.** Computers & Security, Seoul, v. 106, 2021.

[206] LOPES F.; AGNELO J.; TEIXEIRA C.A.; LARANJEIRO N.; BERNARDINO J. **Automating orthogonal defect classification using machine learning algorithms**. Future Generation Computer Systems, Coimbra, v. 102, p. 932-947, 2020.

[207] ESNAASHARI M.; DAMIA A.H. **Automation of software test data generation using genetic algorithm and reinforcement learning**. Expert Systems with Applications, Tehran, v. 183, 2021.

[208] PANDEY S.K.; MISHRA R.B.; TRIPATHI A.K. **BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques.** Expert Systems with Applications, Varanasi, v. 144, 2020.

[209] JIN C. **Cross-project software defect prediction based on domain adaptation learning and optimization.** Expert Systems with Applications, Wuhan, v. 171, 2021.

[210] DURMAZ E.; TÜMER M.B. **Intelligent software debugging: A reinforcement learning approach for detecting the shortest crashing scenarios**. Expert Systems with Applications, Istanbul, v. 198, 2022.

[211] YUCALAR F.; OZCIFT A.; BORANDAG E.; KILINC D. **Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability**. Engineering Science and Technology, an International Journal, Manisa, v. 23, p. 938-950, 2020.

[212] CAI G.; SU Q.; HU Z. **Automated test case generation for path coverage by using grey prediction evolution algorithm with improved scatter search strategy**. Engineering Applications of Artificial Intelligence, Jingzhou, v. 106, 2021.

[213] SU Q.; CAI G.; HU Z.; YANG X. **Test case generation using improved differential evolution algorithms with novel hypercube-based learning strategies.** Engineering Applications of Artificial Intelligence, Jingzhou, v. 112, 2022.

[214] TSAI, Wei-Tek; ZHANG, Li; HU, Shufeng; FAN, Zizheng; WANG, Qianyu. **Crowdtesting Practices and Models: An Empirical Approach**. Information and Software Technology, Beijing, v. 154, 2023.

[215] YANG, Xin-She. **Genetic Algorithms**. In: YANG, Xin-She (Ed.). Nature-Inspired Optimization Algorithms (Second Edition), London, Academic Press, 2021. cap. 6, p. 91-100.

[216] GRUETZEMACHER, Ross. **The Power of Natural Language Processing**. Harvard Business Review, 2022. Disponível em: https://hbr.org/2022/04/the-power-of-natural-language-processing. Acesso em: 16 abr. 2023.

[217] MAYEDA, M.; ANDREWS, A. **Evaluating software testing techniques: A systematic mapping study**. In: Advances in Computers. Denver, 2021. v. 123, p. 41-114.

[218] INDOLIA, S.; GOSWAMI, A. K.; MISHRA, S. P.; ASOPA, P. **Conceptual Understanding of Convolutional Neural Network - A Deep Learning Approach**. Procedia Computer Science, Rajasthan, 2018. v. 132, p. 679-688.