International Journal of Applied Information Systems (IJAIS) ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 13 - No. 1, August 2025 - www.ijais.org

Exploring Search-based Applications in the Software Development Life Cycle: A Literature Review

Abeer Alarainy
Imam Mohammad Ibn Saud Islamic University
Information Technology
Riyadh
Saudi Arabia

Nora Madi
King Saud University
Software Engineering
Riyadh
Saudi Arabia

Aljawharah Al-Muaythir
Alfaisal University
Software Engineering
Riyadh
Saudi Arabia

Abir Benabid Najjar
King Saud University
Software Engineering
Riyadh
Saudi Arabia

ABSTRACT

Search-Based Software Engineering (SBSE) research is having a tangible impact on the wider Software Engineering (SE) community. SBSE tackles SE problems by reformulating them into search problems and then uses heuristic techniques to discover optimal or sub-optimal solutions. Despite the success of SBSE, it faces several challenges, such as the need for rigorous evaluation of its robustness and scalability, as well as bridging the gap between academic research and practical industrial application. Also, it must address the dynamic nature of software systems by incorporating user needs and preferences into its approaches. Additionally, automating tasks such as algorithm selection and evaluation is critical for improving efficiency and practicality. This study presents a literature review of SBSE research across different Software Development Life Cycle (SDLC) phases by reviewing recent publications from 2019 to 2024. It analyzes studies based on SDLC stages and associated problems, with a focus on the SBSE algorithms employed. Also, it highlights current trends in SBSE research, and identifies gaps for future research directions.

Keywords:

Search Based Software Engineering, Meta-heuristic, Software Engineering, Software Development Life Cycle, Systematic Review

1. INTRODUCTION

The SDLC phases, including Requirement Analysis, Design, Development, Testing, Deployment, and Maintenance, present numerous challenges that require software engineers to make decisions or choices depending on project goals. Examples include selecting and prioritizing requirements, refactoring steps, and optimizing test cases. Nonetheless, project management (PM) also plays a crucial role in the SDLC, presenting optimization challenges such as guiding resource allocation to balance team capacity, budget, and project timelines effectively. These tasks, among others, are considered optimization problems, which can be addressed using SBSE by formulating them as search problems to find optimal or near-optimal solutions. SBSE is an active field of research [1] that leverages metaheuristic algorithms, such as genetic algorithms (GA), particle swarm optimization (PSO), simulated annealing (SA), Genetic Programming (GP), Evolutionary Algorithms (EAs), Dragonfly Algorithm (DA), and Ant Colony Optimization (ACO) to automate and optimize SE tasks. The appeal of SBSE lies in its potential to enhance the SDLC processes by minimizing project duration, cost, and failure rates while improving efficiency, scalability, and overall quality. This literature review examines the current state of SBSE research, with a focus on how these optimization techniques are applied across different SDLC phases and addressing specific problems within each phase.Through this review, the following research questions are addressed:

(1) What problems arise during different stages of the SDLC that can be addressed by the application of SBSE?
(2) What are the current trends in SBSE research? and
(3) What are the open gaps for potential future directions in SBSE?

The key contributions of this study include the following:

—Conducting a literature review of SBSE algorithms applied across various SDLC stages by reviewing recent publications from 2019 to 2024.
—Categorizing primary studies based on the SDLC stage they address.
—Organizing the primary studies by the specific optimization problems they aim to solve within the SDLC.
—Analyzing the primary studies with a focus on the algorithms employed, highlighting commonly used methods in SBSE.
—Summarizing current trends in SBSE research to guide future exploration in this domain.
—Providing insights into less-explored areas and suggesting future research directions, including opportunities to apply SBSE in underrepresented SDLC stages and to explore new or hybrid algorithmic approaches.

The rest of the report is structured as follows: Section II presents the related studies, Section III describes the literature review process followed including research questions, search strategy for primary studies, the selection process, and data extraction details. Section IV analyzes the primary studies, including a quantitative and literature analysis. Section V discusses the answers to the research questions. Section VI is on validity threats. Finally, Section VII concludes and suggests future work.

2. RELATED WORK

This section discusses some related works that explore SBSE in the form of systematic mapping (SM) or systematic literature review (SLR). These papers are focused either on a specific task, area, or technique within SBSE or provide a general

overview of regional or community contributions. Table 1 provides an overview of these studies, listing details like publication year, focus of the study, and type of review. The well-known paper by [2] offers a foundational overview of SBSE, classifying key trends, popular optimization techniques like GA, and applications across the software engineering lifecycle—from requirements and planning to testing and maintenance. It highlights gaps in existing research and suggests potential areas for future work, making it a valuable resource for understanding SBSE's growth and impact. However, being over a decade old, this review doesn't reflect the latest advancements and emerging techniques in SBSE. This study, in contrast, brings an up-to-date perspective, focusing on recent developments and applications. [1] provide a mapping study of the Symposium on Search-Based Software Engineering publications. The aim was to identify trends and evolution of topics within SBSE and to highlight the symposium's role in establishing SBSE's maturity. The evaluation showed that SSBSE research primarily focuses on SE tasks like software testing, debugging, design, and maintenance, with evolutionary algorithms being the most widely used search technique as it was employed in 75% of the papers. While [1] identify trends within SSBSE publications, this work offers insights into SBSE's application, addressing problems specific to each SDLC stage by exploring studies across a number of resources. [3] compile and analyze contributions from the Spanish SBSE research community, examining software engineering tasks, proposed algorithms, and collaborative efforts. The study offers insights into research trends and future directions, highlighting a strong focus on optimization challenges across tasks like planning, design, and testing. It also explores a variety of optimization techniques, ranging from exact search methods to evolutionary computation and swarm intelligence. The paper emphasizes the growth of collaboration within Spain and, as a result, does not examine the broader, international SBSE community. Other studies target specific areas within SBSE. For instance, [4] focus on Multi-Objective Evolutionary Algorithms (MOEAs) in SBSE. Their review categorizes widely used metrics and explores their applications across various SE fields, including requirements engineering, testing, and verification. They found that software testing is the most frequently applied area in SBSE, followed by software design, requirements, management, and verification. This aligns with this study, which also observed that testing is the most prominent area in SBSE research. Similarly, [5] conduct a systematic review of nature-inspired metaheuristic methods specifically within software testing. They categorize and compare algorithms such as GA, ACO, cuckoo search, and artificial bee colony optimization (ABC), emphasizing their applications in automated test case generation, test data optimization, and quality assurance. Their review also outlines key metrics like mutation score, complexity, and scalability for evaluating these algorithms, and discusses their strengths, limitations, and potential for further development. Additionally, they highlight open issues in software testing, including the need for better resource management. The findings indicate that a significant number of selected papers focused on improving efficiency, performance, and reducing time and cost in software testing. However, unlike these focused studies, this research takes a broad approach by examining SBSE applications and the algorithms used across the SDLC. In addition, [6] examine the application of swarm intelligence algorithms like Grey Wolf Optimization (GWO), Whale Optimization Algorithm (WOA), Harris Hawks Optimizer (HHO), and Moth Flame Optimizer (MFO) in SE tasks. Their systematic review details these algorithms' use in software testing, defect prediction, and reliability, while also identifying potential areas for future research, such as software re-modularization and re-engineering. Although the paper identifies certain gaps and potential applications, it does not elaborate on the approach used to determine them. Additionally, this review provides insights beyond swarm intelligence algorithms.

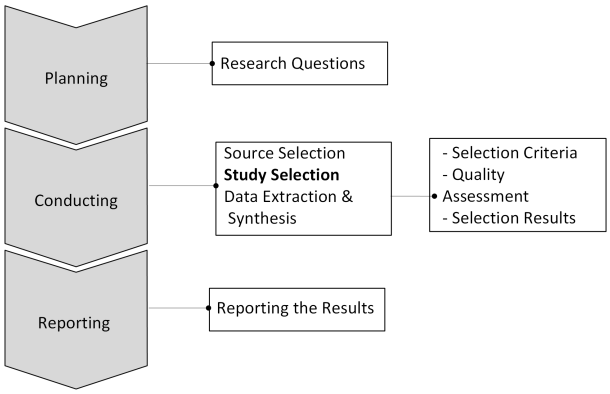

Fig. 1. Literature Review Process Framework.

Most of these papers focus on specific areas or techniques within SBSE. For instance, [5] examine nature-inspired metaheuristics for software testing, [4] focus on MOEAs, [6] review swarm intelligence algorithms applied in software engineering, and [3] explore the contributions of the Spanish SBSE community across various tasks. In contrast, [1] provide a broad mapping study of SBSE research presented at the symposium, covering a range of SE tasks and computational techniques without focusing on any one application or algorithm. However, while SSBSE is a key venue dedicated solely to SBSE, research in this field is also published across other conferences and journals in software engineering. Therefore, this review includes a number of sources and encompasses research from the past six years, 2019 to 2024, providing a recent look at how SBSE is applied to tackle specific problems in the SDLC, while providing future research directions that extend beyond individual techniques or regions.

3. RESEARCH METHODOLOGY

A literature review comprises three main phases: planning, conducting, and reporting the review. The literature review was conducted by following the guidelines of [7] and [8], and the collected data was analysed in an unbiased and structured fashion. The first basic step to start the process of literature review was the planning process. Figure 1 illustrates the flowchart outlining the process that was followed.

3.1 Planning the Literature Review

Three research questions have been identified based on the motivation to perform the literature review. The answers to these questions provide evidence-based support for exploring the application of SBSE in the SDLC. Table 2 outlines the research questions along with their purposes. Additionally, the scope and objectives of the review have been clarified using the PICO criteria (Population, Intervention, Comparison, Outcomes) defined by [7], as shown in Table 3.

3.2 Conductiong the LR

It involves searching electronic databases using the search string and extracts the relevant studies to answer the research questions. This process includes developing a literature search strategy, selecting studies, and conducting data extraction and synthesis.

3.2.1 Literature Search Strategy. The search process involves selecting repositories, defining the search string, and retrieving primary studies from the repositories.

3.2.1.1 Source Selection. The largest and most popular online repositories have been selected, including the ACM Digital Library, IEEE Xplore, Wiley, and SpringerLink. Additionally, a forward snowballing of relevant studies has been included.

Table 1. Summary of Existing Secondary Studies on SBSE.

Title	Focus	Review method	Period-span	Primary papers	Ref.
Search-Based Software Engineering: Trends, Techniques and Applications	Surveys SBSE's application in software testing, requirements, project management, and maintenance. Categorizes research by techniques, problems, and results.	Review	1998-2008	-	[2]
The Symposium on Search-Based Software Engineering: Past, Present and Future	Examines the evolution of SSBSE publications, offering insights to researchers to strengthen the symposium.	SM	2009-2019	134	[1]
A systematic literature review of the SBSE research community in Spain	Compiles and analyzes Spain's research contributions in SE task optimization, novel algorithm proposals, and emerging research interests.	SLR	2001-February 2019	232	[3]
Performance Evaluation Metrics for Multi-Objective Evolutionary Algorithms in Search-Based Software Engineering: Systematic Literature Review	Assesses the performance of MOEAs in SBSE, focusing on commonly used evaluation metrics.	SLR	2000-2020	105	[4]
Nature-inspired metaheuristic methods in software testing	Focuses on the optimization of test case generation using various nature-inspired metaheuristic methods, such as GA, ACO, and others.	SLR	2015-2022	65	[5]
A Systematic Literature Review on Robust Swarm Intelligence Algorithms in Search-Based Software Engineering	Reviews applications of four swarm algorithms (GWO, WOA, HHO, MFO) in areas like software testing, defect prediction, and reliability.	SLR	2014-2022	46	[6]

Table 2. Research questions and their motivation.

NO.	Research Questions	Motivations
RQ1	What problems arise during different stages of the software development lifecycle that can be addressed by the application of SBSE?	Software development faces challenges at various stages, and SBSE offers optimization-driven solutions. The goal is to identify key problems where SBSE can enhance efficiency and scalability, improving software development practices.
RQ2	What are the current trends in SBSE research?	SBSE is a rapidly evolving field, and understanding recent trends helps researchers stay updated on new methods and applications. This question aims to provide an overview of current advancements and emerging hot topics in SBSE utilised in the SDLC.
RQ3	What are the open gaps for potential future directions in SBSE?	There are still many unexplored opportunities and unresolved challenges in SBSE. Identifying open gaps will guide future directions in expanding SBSE's applicability to new areas in software engineering.

3.2.1.2 Search String. Conducted research using only primary sources with the following search string: (Software OR Software development life cycle OR SDLC) AND (requirement OR design OR analysis OR architecture OR implementation OR test OR maintenance) AND (Search-Based Software Engineering OR SBSE) AND (Meta-heuristic OR Genetic Algorithms OR Simulated Annealing OR Particle Swarm OR Ant Colony OR Hill Climbing OR Tabu Search OR Multi-Objective Evolutionary Algorithms OR MOEA) NOT (Machine Learning OR Deep Learning OR Systematic).

3.2.1.3 Search Results and Documentation. Four search strings were applied on four different databases, extracting research papers from the years 2019 to 2024. A full-text search in the research papers resulted in a total of 224 studies retrieved from the repositories. This broad list may include many irrelevant studies.

3.2.2 Study Selection. The selection of primary studies includes the description of selection criteria, the selection process, and the study quality assessment.

3.2.2.1 Selection Criteria. This step is necessary to select only relevant studies and remove irrelevant ones. Inclusion and exclusion criteria, as defined in Table 4, are used for this purpose.

3.2.2.2 Quality Assessment. Only using inclusion and exclusion criteria is not enough to extract the most relevant research studies. The quality of the papers should be evaluated based on the following questions:

(1) Were the research purpose and objectives clearly stated?

Table 3. PICO criteria to define the scope and goal of LR.

Population	The articles and conferences about SBSE in the SDLC.
Intervention	Application of SBSE in the SDLC
Comparison	Analyzing the problems at each stage of the SDLC and evaluating the application of SBSE for each problem.
Outcomes	Overview of issues that arise during the SDLC and the application of SBSE to address these issues, while also identifying gaps for future research directions and highlighting current trends in SBSE research.

Table 4. Inclusion and exclusion criteria.

Inclusion Criteria	Exclusion Criteria
Studies applying SBSE in SDLC	Studies applying Machine Learning (ML)
Studies published within the last 6 years (between 2019-2024)	Papers published before 2019.
Studies published entirely in English	Non-English studies
Peer-reviewed studies	Technical reports, government reports, letters and editorial, short notes, abstract only studies.
Studies identified during the literature review phase were not considered for further analysis.	Duplicate studies have been removed

(2) Was the research problem well-defined?

(3) Does the research include an SBSE algorithm?

(4) Does the research concentrate on a specific SDLC stage?

According to guidelines of quality assessment specified by [7], the questions were scored on a scale of 0 (No), 0.5 (partial), and 1 (Yes). The total score determines the quality of the paper, with a threshold score of 3 set to ensure the quality of primary studies. Only studies with a quality score of 3 or higher were included.

3.2.2.3 Selection Resluts. The study selection process consists of two main steps:
Initial Selection: A total of 224 primary studies were imported into Rayyan, a tool designed to facilitate the review process with a user-friendly interface and a blind screening feature that reduces the risk of bias. During the import, three duplicates were identified. The studies were divided among the authors, with at least two authors reviewing the titles and abstracts to exclude irrelevant literature. After this initial screening, the full texts of the selected studies were examined to ensure they contained relevant information for further analysis. Once the screening was completed, the blind mode was disabled, and the authors agreed to exclude 119 studies. They included 83 studies, classified 5 studies as "maybe," and identified 14 studies with conflicts. The authors then held a meeting to resolve these conflicts and reached a consensus regarding the studies in question. Ultimately, 138 studies were excluded, and 71 studies were included.
Final Selection: The studies that passed the initial selection underwent a quality assessment before being chosen for further analysis. The resulting papers from this final selection served as a seed set for conducting forward snowballing. After a thorough review of these full texts, 71 studies were selected for further analysis. A forward snowballing procedure was then implemented, which led to the identification of an additional 30 studies. Consequently, a total of 101 studies were analyzed using quality assessment criteria, resulting in the exclusion of 16 studies. In the end, 85 studies were deemed ready for data extraction. Figure 2 illustrates the complete process of searching for and selecting the primary studies.

3.2.3 Data Extraction and Synthesis.

3.2.3.1 Data Extraction. The data extraction form was created to ensure the consistency and accuracy of the information collected. This form includes fields that correspond to the research questions as well as publication details. During a consensus meeting, team members discussed and proposed the structure of the form. Table 5 provides information relevant to addressing the research questions. For each paper, the collected data was organized into three categories:

—Publication-related data: digital library, title, year, and type (e.g., journal or conference).

—SBSE-related data: the problem addressed, the optimization algorithm used, and the stage of the SDLC.

—Research-related data: identified trends and suggested future research directions.

3.2.3.2 Data Synthesis. The data synthesis process converts the extracted data into valuable information that answers the research questions. To streamline the analysis, the selected primary studies were organized into a structured format, categorized into three distinct groups: one based on the stage of the Software Development Life Cycle (SDLC), another based on the type of problem addressed, and the third based on the algorithms utilized.

3.2.4 Reporting the Review. In the final stage of a literature review, researchers conduct a comprehensive analysis of the results related to the research questions. This in-depth examination enables synthesizing existing research findings and outlining directions for future research initiatives. The review and discussion are presented in the analysis and discussion sections.

4. ANALYSIS

This section presents an analysis of the primary studies in two parts: publication analysis, which provides qualitative data on the selected studies, and literature analysis, which reviews the research discussed in the papers.

4.1 Publication Analysis

Figure 3 shows that the majority of SBSE-related publications come from IEEE, with a total of 76 publications, significantly surpassing those from other publishers. Figure 4 indicates that the SBSE field is dynamic, with significant contributions from both publishing avenues. Data shows that 56% of SBSE publications are from conferences, while journals account for 44%.

4.2 Literature Analysis

This section reviews the studies related to the specific stages of the SDLC where problems have been identified. Table 6 illus-

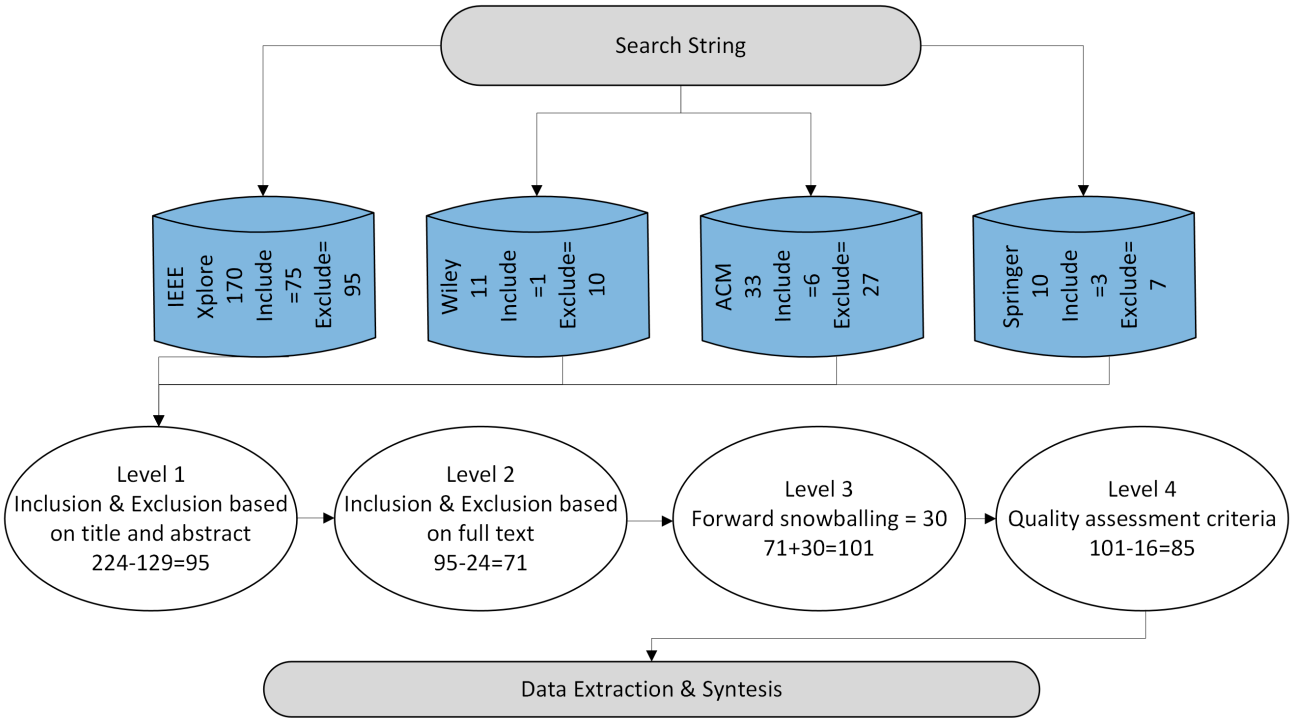

Fig. 2. Study selection process.

Table 5. Data Extraction Form.

ID	Data Item	Description	Relevant RQ
D1	Title of paper	What is the article's title?	Manuscript Information
D2	Year of publication	Year the study was published	Manuscript Information
D3	Digital Library	Name of repository (e.g, IEEE, ACM,)	Manuscript Information
D4	Type	Type of article such as journal or conference	Manuscript Information
D5	SBSE Problem Addressed	Specific problem in SDLC addressed by SBSE (e.g., test case generation, SPS).	RQ1
D6	SBSE Algorithm Used	Type of SBSE algorithm applied (e.g., ACO, PSO).	RQ1
D7	SDLC Stage	The phase(s) of SDLC where SBSE was applied (e.g., testing, design).	RQ1
D8	Trends Identified	Current trends observed in SBSE research.	RQ2
D9	Future Research Directions	Proposed directions for further research.	RQ3

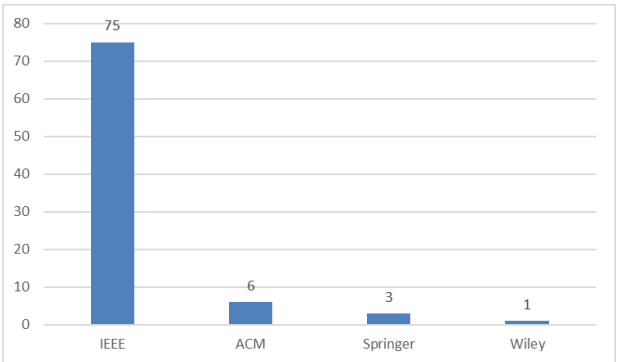

Fig. 3. Distribution of Publications by Publisher.

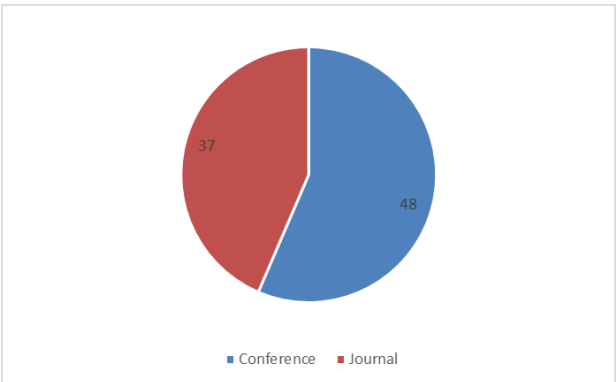

Fig. 4. Distribution of Selected Studies by Publication Venue.

trates the frequency of these problems at each stage. The section is organized according to the most frequently reported issues, which include testing, maintenance, design, project management, and deployment.

4.2.1 Testing. The testing phase reveals the highest frequency of issues, with 55 studies indicating that this stage is where most problems arise and where SBSE methods are frequently applied.

Table 6. Summary of Optimization Problems, SDLC Stages, Applied SBSE Algorithms, and References.

Problem	Freq.	SDLC Stage	SBSE Algorithm	Reference
Test Case Generation and Selection	18	Testing	DynaMOSA, NSGA-II, ACO, EAs, Hybrid (HWOA+ ACO), TLBO, Hybrid (Iterated Local Search + Tabu List), CSA, U-NSGA-III, MAEO, GA, Bi-objective Dragonfly Algorithm (BDA), DeepEvolution, Multiple black hole (MBH), Self-Contained Hierarchical Optimization Protocol, Hybrid (PSO + Firefly Cuckoo Search (FCSA))	[9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26]
Test Data Generation	14	Testing	Hybird (BSA+SA), GA, Multi-population genetic algorithm (MGA), Hybrid (ACO + Negative Selection Algorithm (NSA), Intelligent Optimization Algorithms, Coevolutionary Genetic Algorithm (CGA), Hybrid (GWO + GA), DynaMOSA, Hybrid (GA + immune inspired algorithm), Shuffled Frog Leaping Algorithm (SFLA)	[27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40]
Refactoring	9	Maintenance	NSGA-II, GA, Hybrid (Multi-objective search + Clustering algorithms), Multi-objective Search	[41], [42], [15], [43], [44], [45], [46], [47], [48]
Software Project Scheduling Problem	6	PM	Bi-population discrete evolutionary algorithm (IFBPD-EA), MOEA, EAs, Greedy and Parallel Scheduling (GPS), PSO	[49], [50], [51], [52], [53], [54]
Path Selection and Code Coverage	4	Testing	DynaMOSA, GA, GP, DA	[55], [56], [57], [58]
Bug Detection	3	Testing	DynaMOSA, Novel strategy (SBSTML), NSGA-II	[59], [60], [61]
Bug Localization	1	Maintenance	Interactive SBSE	[62]
Modularization and Re-Modularization	3	Design	NSGA-II, GA, Reinforcement Learning-based Iterated Local Search	[63], [64], [65]
Test Suite Generation and Optimization	3	Testing	WOA, DynaMosa, NSGA-II	[66], [67], [68]
Multi-Objective Testing Resource Allocation Problem (MOTRAP)	2	Testing	MOEAs	[69], [70]
Community Smells Detection	1	Maintenance	GP	[71]
Detecting Code Smells	1	Maintenance	Weighted Cockroach Swarm Optimization (WCSO)	[72]
Discovering Metamorphic Relations	1	Testing	PSO	[73]
Fault Localization	1	Testing	Jaya Algorithm	[74]
Minimizing Problematic Couplings Within the Software Architecture	1	Design	NSGA-II	[75]
Optimal Deployment	1	Deployment	SA	[76]
Optimal Software Architecture within a complex design space	1	Design	NSGA-II	[77]
Software Effort Estimation	1	PM	Chaotic Particle Swarm Optimization (CPSO)	[78]
Requirements Traceability Link Recovery	1	Maintenance	NSGA-II	[79]
Test Report Prioritization	1	Testing	Hybrid (Greedy Algorithm + GA+ ART)	[80]
Software Mutation Testing	1	Testing	Forrest Optimization Algorithm (FOA)	[81]
Pairwise Testing	1	Testing	Gravitational Search Algorithm (GSA)	[82]
Test Path Generation	1	Testing	ACO	[83]
Test Unit Generation	1	Testing	DynaMosa	[84]
Simulation-based testing for Advanced Driver-Assistance Systems	1	Testing	NSGA-II	[85]
Synthesizing Pareto-optimal policies in Markov decision processes	1	Design	Multi-objective genetic algorithms (MOGAs)	[86]
Enhance the greenness of software	1	Design	Genetic Improvement (GI)	[87]
Software Clustering	1	Maintenance	Hybrid (GA+ Krill Herd (KH))	[88]
Identification and Evaluation of Microservices from User Stories	1	Design	GP	[89]
Test orders in service-oriented architecture	1	Testing	GA	[90]
Automate crash reproduction	1	Testing	Guided Genetic Algorithm (GGA)	[54]
Predictions of fault slip-through	1	Testing	Cat Swarm Optimization (CSO)	[91]

Numerous problems are addressed during this phase, particularly in Test Case Generation and Selection, which are the focus of 18 studies ([9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], and [26]).

One notable contribution to the field is the introduction of MoesART, an innovative adaptive random testing algorithm based on multi-objective evolutionary search. This approach, detailed in paper [10], aims to enhance test case diversity and improve the effectiveness of failure detection. MoesART incorporates three diversity perspectives: dispersion, balance, and proportionality, while utilizing the NSGA-II framework to select optimal test cases. The findings suggest that considering multiple diversity objectives significantly enhances the failure detection capabilities of the test cases. Additionally, [14] presents InSPeCT, a novel Path Condition solver that employs Iterated Local Search (ILS) with a Tabu List. This method addresses the challenge of slow automated test case generation in software testing, which is largely due to the high computational complexity associated with Satisfiability Modulo Theories (SMT) solvers used to resolve Path Conditions (PCs). Moreover, [19] introduces a new t-way testing strategy utilizing a Bi-objective Dragonfly Algorithm (BDA) to generate prioritized test suites. This approach addresses the limitations of existing methods that focus primarily on test case weights without considering their priority. The BDA simultaneously optimizes both test case weight and priority, demonstrating competitive performance against traditional t-way strategies regarding test suite size and prioritization. Furthermore, [21] presents a novel variant of the black hole optimization algorithm, called binary multiple black hole (BMBH) optimization, aimed at enhancing the efficiency of combinatorial searching-based software testing (CSST). BMBH utilizes multiple swarms to improve solution exploration and reduce the number of required test cases, achieving reduction rates of over 60% for certain problems. In addition, [24] introduces SCHOP, a novel approach that integrates seeding and constraint support within a harmony search algorithm to optimize test suite size in combinatorial t-way testing. SCHOP addresses challenges in test case generation, enhancing software quality while managing combinatorial explosion. Finally, [25] presents a hybrid approach for software testing that combines PSO, Bee Colony Optimization (BCO), and Firefly Cuckoo Search Algorithms (FCSA) to improve model-based testing. This approach aims to optimize time and cost in the software testing process while ensuring efficient automatic test case generation and execution.

The second identified problem is Test Data Generation, which has been addressed by 14 studies. [42], [30], [31], and [40] utilize GA to tackle this issue. [28] presents BackIP, a novel hybrid approach for mutation-based test data generation that combines the Backtracking Search Optimization Algorithm (BSA) and Integer Programming. This approach aims to enhance the efficiency of test data generation and reduction while maximizing coverage metrics and minimizing the occurrence of equivalent mutants. Additionally, [33] introduces a novel Hybrid ACO-NSA approach for automated test data generation in Java environments. This method integrates ACO and the Negative Selection Algorithm (NSA) to improve software testing efficiency. The performance of the Hybrid ACO-NSA technique is evaluated using metrics such as Average Coverage (AC), Average Generations (AG), Average Time (AT), and Success Rate (SR). These metrics demonstrate the efficiency and effectiveness of the proposed approach in generating automated test data for Java environments. Furthermore, [36] presents a hybrid method named Fuzuli, which combines the GWO algorithm with GA to automatically generate optimal test data for software structural testing. The primary objectives of Fuzuli are to enhance branch coverage, improve the success rate of test data generation, and increase both stability and speed. Finally, [38] introduces a novel method using the Shuffled Frog Leaping Algorithm (SFLA) for automatic software test data generation. This study addresses the challenges of cost and time in software testing while emphasizing the importance of branch coverage as a fitness function. It demonstrates the effectiveness of SFLA in generating efficient test data with a high convergence speed.

The third and fourth most frequent problems in software testing are path coverage and bug detection. Path coverage has been addressed in various studies [55], [56], [**?**], and [58], using techniques such as GA, DynaMOSA, GP, and PSO. In contrast, bug detection has been tackled with more innovative approaches. For example, [59] introduces a novel method called Defect Prediction Guided Search-Based Software Testing (SBSTDPG), which integrates defect prediction with search-based software testing (SBST) to improve bug detection efficiency. This method employs a budget allocation algorithm based on defect scores to prioritize testing in areas of code that are likely to be defective. Empirical evaluations show that SBSTDPG identifies significantly more bugs compared to traditional methods, particularly in resource-constrained environments, achieving an average of 13.1 additional bugs found within a tight time budget. Furthermore, [60] enhances the bug detection capability of search-based software testing by incorporating defect prediction information. It proposes two innovative approaches: one that allocates time budgets to classes based on their likelihood of being defective (SBSTCL) and another that directs the search algorithm towards these defective areas (SBSTML). Various metrics are utilized to evaluate and compare the performance of defect prediction models, including recall, precision, accuracy, Matthews Correlation Coefficient (MCC), and Area Under the Curve (AUC). The effectiveness of defect predictors is assessed based on their ability to identify buggy areas in software, which is essential for guiding developers in efficient testing. Additionally, [61] presents a new approach called Multi-Objective Crowd Worker Recommendation (MOCOM), designed to optimize the recommendation of crowd workers for crowdsourced testing tasks. This approach aims to maximize bug detection while minimizing costs. MOCOM characterizes crowd workers based on their testing context, capabilities, and domain knowledge, significantly outperforming five state-of-the-art baselines in experimental evaluations.

The fifth and sixth most common issues discussed in the literature are test suite generation and optimization, as highlighted in papers [66], [67], and [68], along with the multi-objective testing resource allocation problem explored in papers [69] and [70]. [66] focuses on improving reusability in software product line (SPL) testing by proposing four reusability metrics: Test Suite Reusability regarding Requirements (TSRR), Test Case Reusability regarding Requirements (TCRR), Test Suite Reusability regarding Configurations (TSRC), and Test Case Reusability regarding Configurations (TCRC). The aim is to enhance reusability through an SBST approach, particularly by employing the NSGA-II algorithm to optimize existing test suites. This study emphasizes the importance of measuring and improving reusability metrics, especially TSRR and TCRR. Additionally, [67] introduces a novel strategy for generating combinatorial test suites using the WOA, addressing the challenges associated with exhaustive testing in software systems. It highlights the effectiveness of t-way testing, which reduces the number of test cases by focusing on the interactions among parameters. Finally, [68] presents a hybrid framework that combines the Firefly Algorithm (FA) and the Differential Evolution Algorithm (DE) to automate test suite generation in model-based testing of object-oriented programs. This approach utilizes UML behavioral state chart models to develop optimized test suites aimed at achieving complete transition path coverage. Experimental results demonstrate that the hybrid FA-DE algorithm significantly outperforms the individual FA and DE algorithms in terms of time complexity and the efficiency of test case generation. [69] and [70] both utilize MOEA to address the multi-objective testing resource allocation problem (MOTRAP). They focus on simultaneously optimizing system reliability, testing cost, and testing time us-

ing different models. [69] proposes a model with a pre-specified reliability constraint and develops enhanced constraint handling techniques (ECHTs), while [70] proposes an architecture-based model (ABM).
The NSGA-II was employed to tackle various issues. In [85] investigates the effectiveness of SBST for Advanced Driver-Assistance Systems (ADAS) by comparing results from two simulators: TASS/Siemens PreScan and ESI Pro-SiVIC. The study reveals that while SBST can generate critical test scenarios in both simulators, there are notable discrepancies in safety violations and the dynamics of vehicles and pedestrians. The findings suggest that future Verification and Validation (V&V) processes should incorporate multiple simulators to enhance robustness and reduce dependency on the specific internal workings of any single simulator.

4.2.2 Maintenance. Maintenance is the second stage after testing in terms of the number of studies. Software maintenance is the process that improves the reliability and stability of software products. Its idea relies on modifying a software system after it has been released to improve performance, fix bugs, add new features to meet evolving user requirements, or adapt to a changed environment. Following design principles during development leads to high-quality software products that are easier and cheaper to maintain. Studies on the maintenance phase focus on multiple issues related to refactoring ([41], [42], [92], [43], [44], [45], [48], [46], [47]) and bug localization [62], along with several other maintenance issues discussed in [71],[72],[79], and [88]. For the refactoring aspect, 11 studies worked on several proposed approaches to address different refactoring issues. [41], the authors proposed DEPICTER approach that helps in refactoring recommendations by representing the design-principle metrics as fitness function using NSGA-II to improve the quality of software design, along with the use of heuristic rules to guide the initialization of the population of potential solutions. Its evaluation of four Java systems demonstrated significant improvements in software design quality. Similarly, 3Erefactor by [42] is a novel automated refactoring approach that addresses architectural inconsistencies using NSGA-II and information fusion techniques. The approach has been validated on six open-source Java projects, showing the effectiveness, efficiency, and executability of 3Erefactor. The experimental results confirm the approach's capability to decrease architectural inconsistencies and improve software quality. Additionally, [92] introduces a novel approach that uses a multi-objective genetic algorithm (MGA) with a penalized fitness function called PMQ (Penalized Modularity Quality) to improve software modularity while minimizing structural disruptions. The evaluation is conducted by employing both single-objective and multi-objective genetic algorithms on eight systems including real-world systems and randomly generated systems. The results demonstrate that the multi-objective approach yields better results in most scenarios, achieving higher modularity quality with fewer changes compared to the single-objective method. To address the limitations in traditional refactoring techniques such as neglecting developer priorities and failing to adapt to ongoing code evolution, [43] introduced a novel interactive approach for software refactoring that combines innovization (innovation through optimization) and the use of NSGA-II. The approach also includes the use of an implicit exploration of the Pareto front of non-dominated solutions to eliminate the need to manually explore the Pareto front for the best trade-offs. This approach facilitates adapting refactoring solutions based on developer feedback while considering other code changes concurrently made by the developer. Based on the implementation of a benchmark of eight open-source systems and two industrial projects, the results support the claim that the proposed approach is more efficient, on average, than existing refactoring techniques. Similarly, DOIMR (Decision and Objective Interactive Multi-Objective Refactoring) approach [45], which considers the developers preferences by helping them to explore both the effects of different refactoring options on code quality and where those refactorings would occur in the code. The tool uses a clustering algorithm to group similar refactoring solutions to make exploration and selection easier for the developer. The paper evaluated the tool with 35 developers and found that it was effective at helping them quickly and easily find relevant refactorings that met their needs.
Also, [47], authors used GA and NSGA-II to evaluate their proposed novel approach. The approach aims to extract developers' knowledge and preferences to find good refactoring recommendations. They combined the use of multi-objective search, clustering, mono-objective search, and user interaction in their approach to facilitate decision-making. Ordering dependencies among refactorings is a concept introduced by [44]. Authors in this work developed an algorithm using GA to detect ordering dependencies and a tool named DPRef which leverages the algorithm's output to generate visual representations of these dependencies as refactoring graphs. It demonstrated high precision in detecting dependencies across 9,595 projects. These advancements collectively simplify and improve refactoring processes. In educational contexts, [46] employ the use of a multi-objective search-based refactoring in Scratch Programs to improve readability and learning outcomes. The approach was tested on 1,000 projects, showing significant enhancements in program comprehension. Also, in the security context, [48] use NSGA-II to balance security and quality attributes in their proposed security-aware refactoring framework. This framework finds trade-offs between improving software quality and maintaining security. For bug localization, a hybrid fitness function in [62] combines human evaluation with automated simulations to improve bug localization in video game models. Tested on an industrial video game with 29 developers, this approach reduced manual effort and missed bugs. It achieved notable improvements in precision, recall, and overall performance metrics. Both papers, [72] and [71] use automated approaches to detect suboptimal patterns in software development, although they target distinct types of problems which are code smells [72] and community smells [71]. [72], this paper focuses on detecting code smells, which are structural characteristics in source code that may cause deeper problems and affect software quality. The paper presents the Weighted Cockroach Swarm Optimization (WCSO) algorithm for automatically identifying code smells. The study shows that WCSO outperforms the other evolutionary algorithms in detecting the targeted code smells across various open-source applications. [71], this paper shifts the focus to community smells, which are organizational and social patterns within software development communities that can negatively impact project success and software quality. The paper introduces the Genetic Programming-based Ensemble Classifier Chain (GP-ECC) approach to detect community smells. GP-ECC's performance is compared against several other multi-label learning (MLL) techniques, and the findings show that GP-ECC achieves superior performance compared to the benchmark methods. [88] and [79], focus on different areas to improve software development processes. [88] introduces a novel algorithm for software clustering, while [79] investigates using a well-known optimization algorithm for requirements traceability link recovery. For software clustering, GAKH algorithm [88] combines Genetic Algorithm (GA) and Krill Herd (KH) algorithms to deliver superior clustering quality. This approach was evaluated on ten different software systems by using TurboMQ demonstrating high-quality clusters. In requirements traceability, NSGA-II [79] paired with information retrieval methods to overcome the advantages of manual creation and maintenance of traceability links reducing manual effort and potential for errors. Based on the conducted evaluation, the approach demonstrates superior performance compared to basic IR techniques. While these studies use and explore different approaches to address distinct chal-

lenges in software engineering, they highlight the effectiveness of using optimization algorithms, specifically SBSE algorithms to improve software development processes and enhance software quality, comprehension, and maintainability.

4.2.3 Design. The works by ([75], [63], [30], [77], [86], [87], [89]) discuss various techniques for software modularization and microservice decomposition, illustrating the importance of automated approaches along with the use of optimization algorithms. Some studies focus on enhancing the modularization methods, while others focus on the challenges associated with transitioning monolithic applications to microservice architectures. For the modularization aspect, [63] suggests a novel approach for software modularization that merges search-based and hierarchical techniques. The suggested approach minimizes the search space by initializing the modularization using a genetic algorithm, and the final modularization is then carried out using a hierarchical algorithm. This combination strategy preserves a hierarchical representation of the software for easier understanding while enhancing the quality of modularization. Additionally, [64] suggests a novel Reinforcement Learning-based Iterated Local Search (RL-ILS) technique for software modularization. ILS consists of two key components: a local search approach and a perturbation method. Finding high-quality modularizations is the aim of RL-ILS. The effectiveness of RL-ILS is demonstrated through experiments on eleven real-world software systems, such as search-based, hierarchical, greedy, and non-hierarchical algorithms, in terms of modularization quality. In order to improve software architecture reliability which is important while designing an architectural structure, [77] investigates the application of genetic programming techniques and (NSGA-II). The proposed method enhances software reliability by identifying architectural tactics, mapping components to these tactics, and optimizing the architecture to balance cost and reliability. A case study on a business reporting system demonstrates the approach's effectiveness in generating optimal architectures that achieve this balance. [86] employs MOEAs to synthesize Pareto-optimal policies for Markov Decision Processes (MDPs) to meet complex quality-of-service (QoS) requirements. The study presents the EvoPoli framework, which uses evolutionary algorithms to search for Pareto-optimal policies that represent trade-offs between different objectives. A case study using an autonomous underwater vehicle (AUV) operating in an ocean environment is used to assess EvoPoli. The experimental findings show that EvoPoli can successfully synthesize policies that balance conflicting objectives, such as maximizing the gathering of scientific data while minimizing mission risk and energy consumption. To optimize software architecture sustainability, [87] suggests integrating GI into the architectural design process to automatically optimize software for reduced energy consumption without compromising performance. The proposed framework integrates sustainability into design, optimization, and continuous evolution phases, automating the creation of energy-efficient systems. The authors suggest future work includes refining algorithms, developing sustainability metrics, and validating the approach through case studies, advancing sustainable software engineering. Also, [75], the authors describe a novel, search-based algorithm based on NSGA-II for automating software isolation. The main idea is to frame isolation as an optimization problem, where the goal is to minimize problematic couplings that cross the boundary of the user-defined portion of code to be isolated. The approach successfully reduces problematic couplings by over 87% on average across various open-source projects. In addition, [89] introduces the Microservices Backlog (MB) model, a semi-automatic approach using genetic programming to optimize microservice granularity at design time, based on user stories. By considering metrics like coupling, cohesion, granularity, semantic similarity, and complexity, MB enables graphical analysis of architectures, which helps identify critical design issues. It outperformed state-of-the-art methods, achieving less coupling, higher cohesion, reduced complexity, and fewer dependencies.

4.2.4 Project Management. Studies on the PM phase have addressed issues related to Software Project Scheduling (SPSP) ([49], [50], [51], [52], [53], [54]) and software effort estimation [78]. Notably, the majority of studies related to PM focused on SPSP. [49] introduces a new mathematical model for improving project scheduling, considering communication costs and team relationships, which are often overlooked in traditional scheduling techniques. To solve the model, the algorithm, called the enhanced Bi-Population Discrete Evolutionary Algorithm (IFBPD-EA), uses feedback mechanisms for adaptive tuning and improved local search strategies. Experimental results show that it outperforms four state-of-the-art algorithms, achieving improvements in project duration and cost reduction. This suggests the algorithm's potential for enhancing scheduling performance, especially in projects with dynamic human interactions and evolving skill requirements. Similarly, [51] examines the disruptive effects of employment turnover. It presents a MOEA that balances the reassignment of existing team members with the hiring of new employees with the necessary skills in order to maximize resource allocation. In tests across dynamic benchmarks and real-world settings, the model showed a 9% reduction in project costs when compared to traditional methods. The study provides insightful information for project managers and future research by demonstrating how efficiently controlling turnover can lower costs, sustain productivity, and enhance project outcomes. Furthermore, [50] presents a novel method for multi-round procurement auctions. The objective is to assist participants in making better choices in complex bidding situations where they could otherwise prioritize personal benefits over the success of the project as a whole. To determine the Nash equilibrium—a balance point that makes bidder selection clear and promotes long-term collaboration—the researchers created a game-theoretic model. The PSO algorithm was very successful at achieving this equilibrium after being fine-tuned, beating alternative techniques in terms of speed and quality. The findings of the study demonstrate that this strategy produces win-win situations by striking a balance between timeframes, expenses, and profitability for bidders and investors, eventually fostering a more collaborative procurement process.
In order to improve project schedules and costs, [52] presents a novel model for SPSP that takes into account the evolution of employees' experiences and learning skills. This method takes into consideration how people develop over time, which results in notable increases in productivity, in contrast to traditional models that frequently ignore human factors. When the model was tested on 24 problem cases, it was found to reduce project length by 40% while maintaining budget. Additionally, the model was compared to six top algorithms; in 63% of the cases, the bi-criterion evolution (BCE) algorithm performed better than the others. These results highlight how crucial it is to factor employee experience into project scheduling and give project managers useful advice on how to allocate resources efficiently in changing settings. [53] also addresses a dynamic situation but in terms of integrating new employees mid-project. By framing SPSP as a multi-objective optimization problem, the model aims to balance project duration and costs while adapting to changing project requirements and employee availability. The proposed heuristic, hNEA, was tested across 18 benchmark scenarios and 3 real-world cases, showing strong performance in managing dynamic events without sacrificing efficiency. The research highlights the importance of flexibility in project management and offers practical insights for managers handling complex, evolving projects. [78] attempts to solve software effort estimation by employing a chaotic Particle Swarm Optimization technique to optimize Use Case complexity weights. Traditional complex-

ity weight levels run the danger of producing inaccurate project planning because they frequently result in sudden classifications and unreliable estimations. Using a Bernoulli chaotic map, the suggested chaotic PSO method produced the best results, exceeding standard PSO in accuracy by a wide margin (p ¡ 0.05). This research fills a gap in optimizing the Use Case Points (UCP) framework, offering practical value for software project managers by enabling more accurate cost and effort estimates. It is recommended that this strategy be extended in future studies using more datasets for validation.

5. DISCUSSION

This section provides answers to the RQs by analysing the selected studies.

RQ1: Addressing SDLC Problems with SBSE Applications

To answer the question, the text is divided into three parts:

Part 1: SBSE Problems

After analyzing 85 primary studies, the problems that arose during the SDLC stages were extracted. Figure 5 illustrates the number of studies that focus on each problem. Test Case Generation and Test Data Generation are the most frequently occurring problems. This highlights a strong emphasis on SBSE within testing-related tasks, likely due to the complexity and resources required for effective testing. Refactoring receives moderate attention which reflects the significance of SBSE in enhancing code quality and maintainability. The Software Project Scheduling Problem (SPSP) appears 6 times, underscoring the importance of SBSE in optimizing project timelines and resource allocation. Path Selection and Coverage is also represented with 4 occurrences, likely due to its role in ensuring comprehensive test coverage. Less common problems addressed by SBSE include Bug Detection as well as Modularization, Re-Modularization, and Test Suite Generation and Optimization, each having a limited but noteworthy focus in research. The Multi-Objective Testing Resource Allocation Problem, Community Smells Detection, Code Smells Detection, Bug Localization, and the discovery of Metamorphic Relations are mentioned once or twice, suggesting they are lacking sufficient focus. Other problems are detailed in Tabel 7.

Part 2: SBSE algorithms used to tackle the problems

Figure 6 illustrates the frequency of the most commonly used algorithms for addressing the SBSE problem. Among the listed algorithms, NSGA-II and the GA are the most commonly used, likely due to their multi-objective capabilities. Other algorithms such as DynaMOSA, PSO, and MOEAs exhibit moderate usage, each being mentioned around 4 to 5 times. Methods like GP, EAs, DA, and ACO are cited less frequently, with 2 to 3 occurrences each. The least represented algorithms, with minimal mentions, include CSO, SA and the WOA. Additionally, the ”Other” category, which encompasses 20 occurrences, includes novel approaches based on SBSE as well as rarely used algorithms such as the Immune Algorithm, Tabu Search, Cockroach Swarm Optimization, FOA, and Cuckoo Search Optimization.

Part 3: SDLC stages where SBSE is used

Figure 7 illustrates the frequency of problems arising at various stages of the SDLC that could potentially be addressed using SBSE. The testing phase presents the highest frequency of issues, with 55 studies indicating that this stage is where most problems occur, making it a prime target for SBSE methods. The maintenance stage follows with approximately 15 studies, suggesting it also faces significant challenges that can be addressed by SBSE, likely due to the complexities involved in updating and improving existing software. The design and PM phase experiences a moderate frequency of issues. In PM, six out of seven instances focus on project scheduling, which is essential for effective resource management. This allows teams to complete projects on time and within budget. The remaining instance pertains to effort estimation. In contrast, the deployment stage highlights only one issue, indicating that the application of SBSE to address deployment-related challenges is limited. Overall, the chart suggests that SBSE methods are primarily relevant and potentially most beneficial during the Testing and Maintenance stages of the SDLC, with a lesser, yet still significant, role in the Design and Project Management phases. Other stages, such as Deployment exhibit fewer issues that SBSE could effectively address.

RQ2: Current Trends in SBSE Research

Figure 8 illustrates the distribution of studies over the years. From 2019 to 2022, there was a steady increase in the number of studies, which peaked in 2022. This trend suggests a growing interest in SBSE during this period, likely due to advancements in the field or increased awareness of its benefits in software engineering. The highest number of studies occurred in 2022, indicating that SBSE gained significant attention and application that year. This peak may be linked to improved methodologies, tools, or successful applications that captured public interest. However, from 2022 to 2024, there was a sharp decline in the number of studies, with a notable drop between 2023 and 2024. This decrease could be attributed to several factors, such as a shift in research focus towards newer or alternative methodologies, including ML which is out of the scope of this paper.

In order to know the annual change in using SBSE algorithms, the literature reviewed in this study shows that the hybrid algorithms and (NSGA-II) appear to be the most used algorithms in 2019 as shown in Figure 9. They have been discussed in a number of applications in testing, and maintenance fields, including bug detection [61], refactoring [47], software clustering problems [87], and test case generation [14]. The term ”(Others)” refers to various algorithms that were reported to be used only once or twice during the year; hence, they are not discussed here. In 2020, (NSGA-II), and (GA) continued to be used remarkably and especially the Multi Population Genetic algorithm (MGA) to support refactoring [43], and test case generation [29],[32],[40]. The number of studies has clearly increased in 2021, specifically in (GA), to play an important role in testing and maintenance fields. GA has been utilized to show its effectiveness in maintenance and testing in terms of unit testing to enhance reliability and performance [27], multi path coverage for test data [55], test data generation to maximize code coverage and enhance software testing efficiency [31]. Studies in 2022 reached the highest number and were more diverse compared to previous years as topics related to testing, maintenance, design, project management, and planning were highlighted. NSGA-II and hybrid algorithms have been used more in the field of testing and maintenance such as [45],[36],[37], and [25]. It is worth noting that there have been promising attempts to use NSGA-II in design. For example, [75], an automated refactoring process for isolating software components from complex architectural dependencies has been proposed. In addition, [77], a search-based method for optimizing software architecture reliability using the NSGA-II algorithm and genetic programming has been proposed by extracting reliability tactics through genetic programming and optimizing component allocation with NSGA-II. Also, [78], optimizing Use Case complexity weight parameters in software effort estimation using chaotic Particle Swarm Optimization (PSO) has been discussed as well the use of multi-objective evolutionary algorithm (MOEA) in project management [52]. In the last two years 2023 and 2024, the literature highlights various algorithms such as the hybrid algorithms that have been used in project management, maintenance, and testing [53],[62],[13], and [33]. Furthermore, the use of DynaMOSA [84], MOEA [70], ACO [83], and PSO [23] in testing and MOEA in project management [82] has been proposed. The diversity and advancement

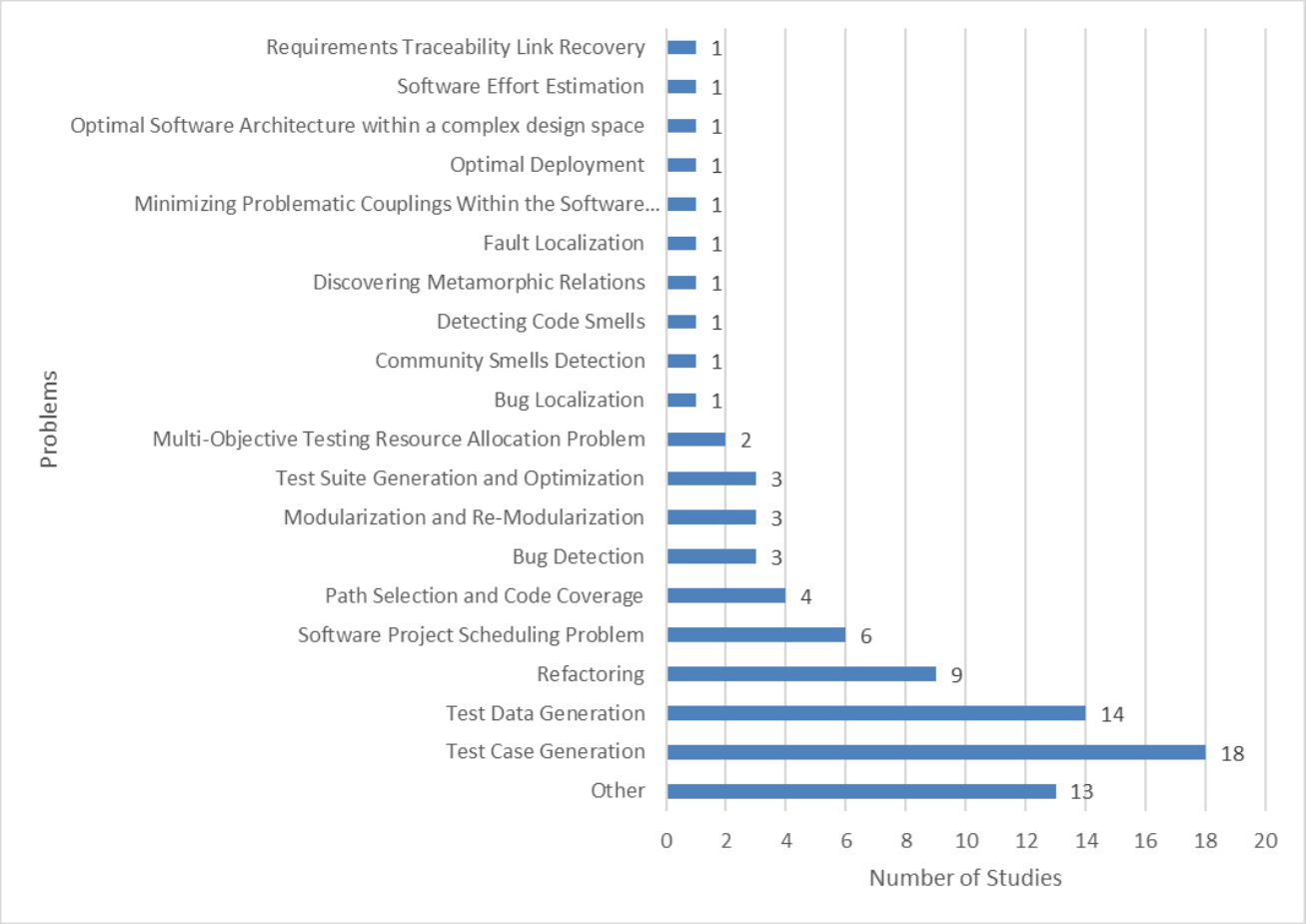

Fig. 5. Number of studies addressing various software engineering problems in the reviewed literature.

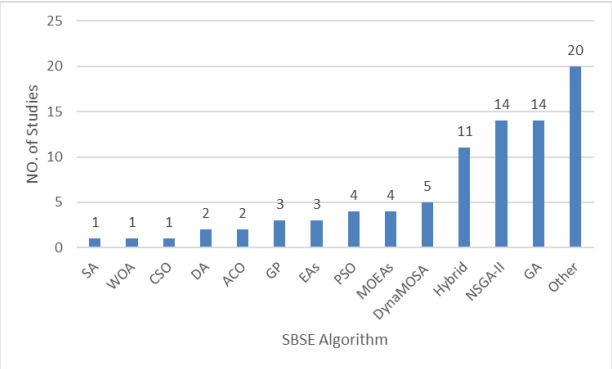

Fig. 6. Frequency of SBSE Algorithms Used in Software Engineering Studies.

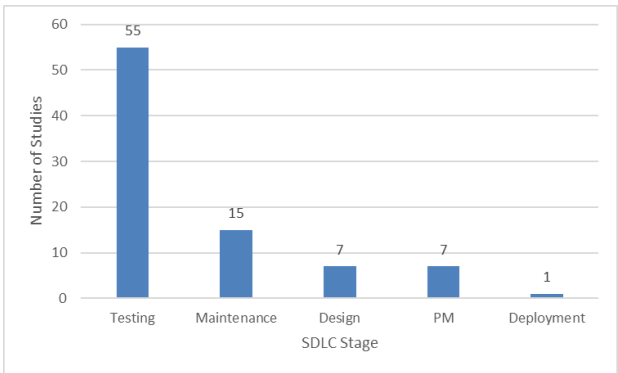

Fig. 7. Distribution of SBSE Studies Across SDLC Stages.

in the use of SBSE algorithms to support and improve SDLC is a strong indicator of their effectiveness and potential to solve a wide range of challenges. This progress encourages continued work on exploring the SBSE algorithms field and highlights the valuable role algorithms play in enhancing and solving SDLC problems.

RQ3: Open Gaps for Potential Future Directions in SBSE

This section responds to RQ3 by analyzing the primary studies and identifying open gaps, pointing to possible directions for further research. In Table 7, the primary studies are analyzed, categorizing them according to the targeted problems and identifying the SBSE algorithms applied to address these problems. Figure 6 presents the number of studies corresponding to each algorithm, including the 'Other' category, which covers 20 articles. Each of these 20 papers uses an algorithm that appeared only once among the 85 analyzed papers. Upon further investigation of the results and variety of these studies, it was found that these algorithms often showed effectiveness, suggesting a valuable area for potential investigation and could lead to broader applications of SBSE methodologies in diverse software engineering contexts. For example, the Greedy and Parallel Scheduling (GPS) algorithm in [54], demonstrates the effectiveness of exploring a less common scheduling technique that takes into account heterogeneous resources with different levels of skill proficiency. Compared to the Parallel Scheduling Scheme (PSS), the GPS algo-

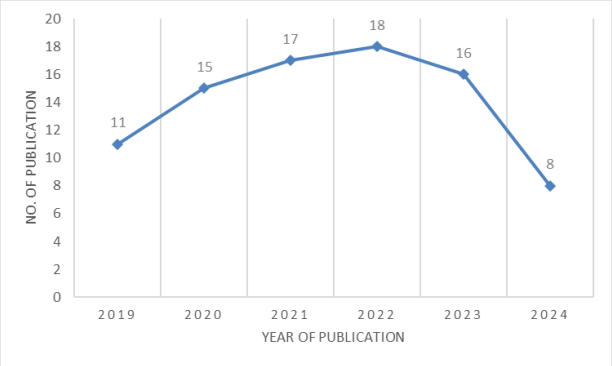

Fig. 8. Trend of SBSE-Related Publications Over the Years.

rithm consistently delivers better performance across different project settings, effectively reducing project duration. Another example is the work by [81], which introduces a new method for software mutation testing that uses a discretized, modified version of the FOA to target and mutate only the most bug-prone paths within a program, thereby reducing mutant creation by about 27.63% compared to traditional techniques. Further investigation was also conducted on algorithms that appeared multiple times in the primary studies. Two bubble charts were created to visualize open gaps: one showing the application of SBSE algorithms to SDLC-related problems (Figure 10) and another illustrating their application across SDLC stages, providing a broader perspective (Figure 11). Figure 10 highlights the versatility of certain algorithms across multiple SDLC optimization problems while also revealing several open gaps. For instance, algorithms like CSO and WOA are less frequently applied to the problems explored. Investigating ways to adapt or broaden the use of these algorithms across a wider range of SDLC challenges could open up new research directions. The NSGA-II algorithm, widely applied across various problems, has not yet been explored in a hybrid approach within the studies, presenting another opportunity for future research. A number of problems, including Software Clustering and Test Report Prioritization, have received limited attention. Future research could examine these problems to determine whether other algorithms might improve effectiveness in solving them. By contrast, Test Case Generation and Selection have received more attention and benefit from a diverse range of algorithms, reflecting their complexity and the need for flexible, more advanced approaches. Test Data Generation, however, has exclusively been approached using GA and hybrid techniques, such as BSA combined with SA and GWO combined with GA. Moreover, some emerging algorithms and recent advances in optimization that employ artificial intelligence and ML are not represented in this study. Exploring these new developments could provide valuable solutions for various SDLC problems in future research. Supporting this is the work by [43], which introduces an interactive tool for software refactoring that uses the multi-objective evolutionary algorithm NSGA-II to generate a set of solutions (i.e. Pareto front). The approach dynamically adapts to developer feedback, providing refactoring suggestions based on real-time code changes to preserve software design quality and minimize the number of recommended refactorings. Evaluation results show the tool's efficiency in comparison to other techniques, demonstrating improvements in the software refactoring process. However, the authors note that exploring the Pareto front is challenging, especially when using an interactive approach. Consequently, they suggest incorporating ML as a potential extension to address this difficulty. In addition, [23] proposes an enhanced version of the PSO algorithm tailored for software test data generation. By introducing modifications, including a new fitness function and balanced exploration and exploitation, this approach overcomes local optima challenges, increasing test coverage while requiring fewer evaluations. This refined PSO method outperforms several existing evolutionary and meta-heuristic algorithms. Although, the authors point out some unresolved issues that need further exploration, suggesting the use of hybrid algorithms or the integration of ML methods to enhance optimization. Only 11 out of 85 studies (approximately 13%) utilized hybrid approaches, suggesting a lower adoption rate than anticipated based on the potential benefits of hybrid methodologies. This suggests another area for future research, as hybrid algorithms could potentially offer robust solutions to SDLC optimization problems where the weaknesses of one approach can be offset by the strengths of another. For example, [33] is a recent study that presents a new approach, the Hybrid ACO-NSA, for automating test data generation in Java environments. The performance of the algorithm surpasses current state-of-the-art-approaches, with a success rate of 99.8%. In this case, the ACO algorithm is good at exploring the search space, while the NSA algorithm is good at finding test data replication, providing complete route coverage, and decreasing the size of a test suite. By combining these two algorithms, the Hybrid ACO-NSA approach is able to generate test data that is both comprehensive and efficient. Another example is Fuzuli [36], which employs both GWO and GA to automatically generate optimal test data for software testing. The algorithm outperforms other algorithms (e.g. PSA and ABC) with an average coverage of %99.98, a success rate of %99.97, and an average output of 2.86. In this algorithm, the GA generates the initial population and applies evolutionary operators, while the GWO directs the search process, resulting in efficient and effective test data generation. From a broader perspective, Figure 11 shows the application of SBSE algorithms across various SDLC stages, including Testing, Design, Maintenance, and Project Management (PM). Design and PM stages have received less attention and less variety of algorithm use compared to Testing. Additionally, none of the surveyed primary studies focused on the Requirements or Development stages. This suggests potential gaps where additional research could investigate the applicability of SBSE algorithms for these stages.

The deployment phase is not represented in the plot, as only a single study addressed this phase—using Simulated Annealing (SA) [76]. That study presents an optimization model for deploying cloud-hosted application components in a way that guarantees multitenancy isolation. The model addresses the challenge of sharing resources among tenants while maintaining isolation to prevent performance degradation due to high workloads. By modelling the deployment problem as a multi-choice multidimensional knapsack problem (MMKP), the authors develop a solution using a simulated annealing-based metaheuristic. The results reveal that the approach achieves near-optimal solutions, balancing isolation needs with resource utilization efficiency, highlighting the potential of metaheuristic optimization in cloud resource management. This indicates an opportunity for further research to explore and expand the use of SBSE algorithms in the deployment phase.

6. THREATS TO VALIDITY

Internal validity: Searches were conducted to identify all relevant studies from four digital libraries. However, it is acknowledged that some studies may have been overlooked, but it is believed that their number will be minimal.

Construct validity: The research questions may not encompass every aspect of the latest advancements in SBSE. To mitigate this issue, brainstorming is employed to effectively pinpoint a set of research questions that adequately address the current research landscape in this study.

Additionally, a potential challenge faced is ensuring the accuracy of the classification of optimization problems within the SDLC. Despite efforts to categorize them as carefully as possible, some

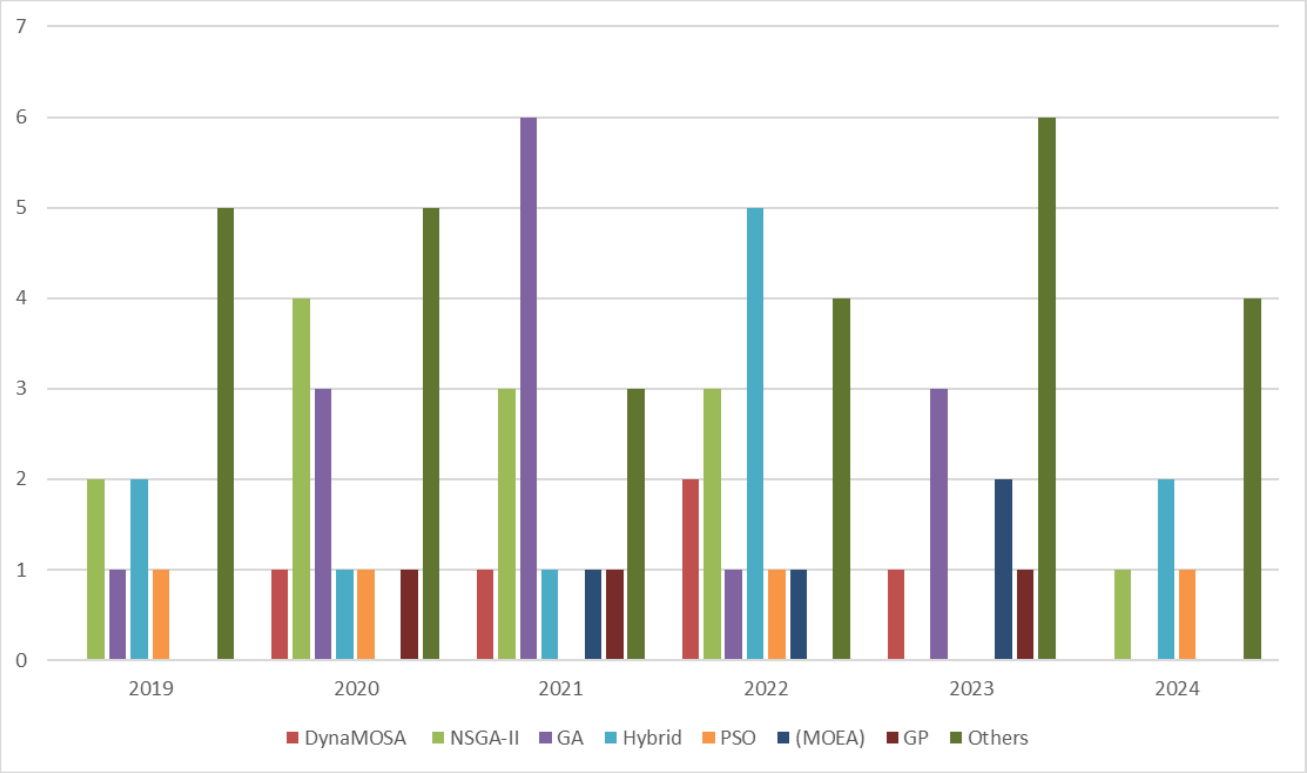

Fig. 9. Yearly Distribution of SBSE Algorithms Used in Studies.

problems might overlap, and the distinction between general and specific classifications isn't always clear.

Conclusion validity: This analysis focuses exclusively on studies that address the use of SBSE within the SDLC. The authors argue that the purpose of this study is to explore and identify potential research directions for the application of SBSE in the SDLC, which can help researchers select relevant areas of application. However, although many studies have utilized ML to address issues that arise during the SDLC, the authors have chosen to exclude it from their research.

Additionally, this study does not evaluate techniques across diverse datasets, as the reviewed primary studies do not consistently report standardized datasets or benchmark scenarios. Future research could explore this direction as more empirical studies with unified reporting become available.

External validity: The findings from this literature review were evaluated in relation to specific studies within the SBSE domain. Thus, the conclusions and classifications are valid only within this particular context. The current study's results are based on qualitative analysis and can serve as a foundation for future research. Additional studies can be examined accordingly.

7. CONCLUSIONS

In this paper, the literature review results on exploring search-based applications in the SDLC are presented. The study confirms the significant interest in applying SBSE algorithms as indicated by the number of reviewed studies. The most common use of SBSE algorithms occurs during the testing phase, where they address various problems, particularly in Test Case Generation and Selection. This observation suggests a potential avenue for researchers to investigate whether the advancements made in testing can be applicable to other SDLC phases, especially in project management, requirements, design, and deployment stages, which have received relatively little attention but are equally important.

Additionally, the work contributes to identifying gaps in current studies, aiding in suggesting future research areas and opportunities. These include exploring the application of SBSE algorithms for enhancement approaches in requirements engineering, development, and deployment challenges, as well as further investigation into hybrid and machine learning approaches.

The goal of this literature review study is to highlight the main research topics related to the application of SBSE in various SDLC phases and associated problems, thereby encouraging both researchers and practitioners to explore the substantial potential that SBSE research areas can offer to support the SDLC as a whole. For future work, there are plans to investigate the applications of machine learning algorithms in addressing SDLC-related optimization problems and compare their effectiveness with traditional SBSE approaches that do not incorporate machine learning.

8. REFERENCES

[1] T. E. Colanzi, W. K. Assunção, S. R. Vergilio, P. R. Farah, and G. Guizzo, "The symposium on search-based software engineering: Past, present and future," *Information and Software Technology*, vol. 127, p. 106372, 2020.

[2] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–61, 2012.

[3] A. Ramírez, P. Delgado-Pérez, J. Ferrer, J. R. Romero, I. Medina-Bulo, and F. Chicano, "A systematic literature review of the sbse research community in spain," *Progress in Artificial Intelligence*, vol. 9, pp. 113–128, 2020.

[4] "Performance evaluation metrics for multi-objective evolutionary algorithms in search-based software engineering: Systematic literature review," *Applied Sciences (Switzerland)*, vol. 11, no. 7, 2021.

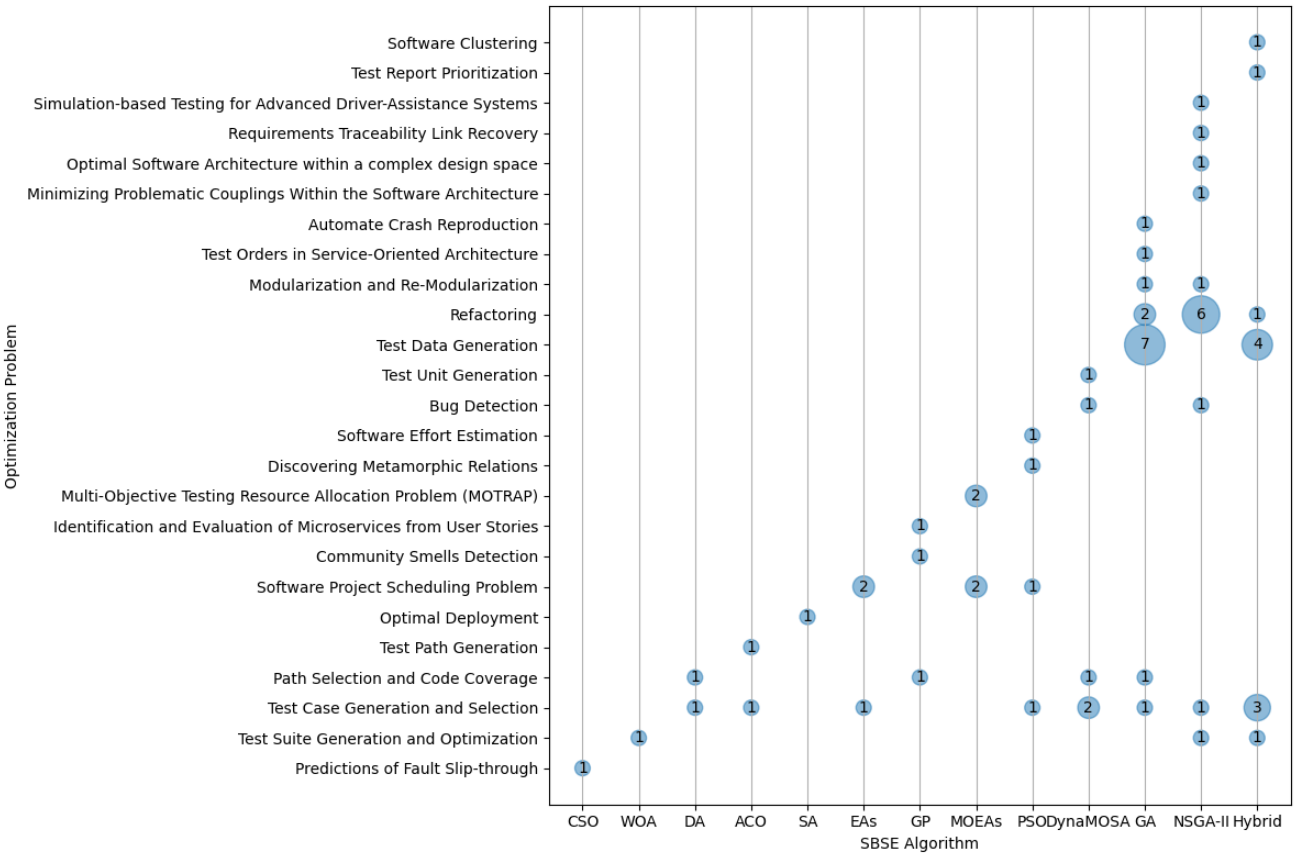

Fig. 10. Mapping of Optimization Problems in SDLC to Applied SBSE Algorithms.

[5] N. Khoshniat, A. Jamarani, A. Ahmadzadeh, M. Haghi Kashani, and E. Mahdipour, "Nature-inspired metaheuristic methods in software testing," *Soft Computing*, vol. 28, no. 2, pp. 1503–1544, 2024.

[6] A. Zeb, F. Din, M. Fayaz, G. Mehmood, and K. Z. Zamli, "A systematic literature review on robust swarm intelligence algorithms in search-based software engineering," *Complexity*, vol. 2023, no. 1, p. 4577581, 2023.

[7] B. Kitchenham and S. M. Charters, "Guidelines for performing systematic literature reviews in software engineering," *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*, no. January 2007, pp. 1–57, 2007.

[8] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th international conference on evaluation and assessment in software engineering (EASE)*, BCS Learning & Development, 2008.

[9] P. Delgado-Pérez, A. Ramírez, K. J. Valle-Gómez, I. Medina-Bulo, and J. R. Romero, "Interevo-tr: Interactive evolutionary test generation with readability assessment," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2580–2596, 2022.

[10] C. Mao, L. Wen, and T. Y. Chen, "Adaptive random test case generation based on multi-objective evolutionary search," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 46–53, IEEE, 2020.

[11] F. Mehboob, A. Rauf, and R. U. R. Qazi, "Evaluating the optimized mutation analysis approach in context of model-based testing," in *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, pp. 1–6, IEEE, 2020.

[12] J. Cao, H. Huang, and F. Liu, "Android unit test case generation based on the strategy of multi-dimensional coverage," in *2021 IEEE 7th International Conference on Cloud Computing and Intelligent Systems (CCIS)*, pp. 114–121, IEEE, 2021.

[13] I. Ghani, W. M. Wan-Kadir, A. F. Arbain, and I. Ghani, "A detection-based multi-objective test case selection algorithm to improve time and efficiency in regression testing," *IEEE Access*, 2024.

[14] F. Chen, A. Gunawan, D. Lo, and S. Kim, "Inspect: Iterated local search for solving path conditions," tech. rep., 2019.

[15] S. Sharma, S. Rizvi, and V. Sharma, "A framework for optimization of software test cases generation using cuckoo search algorithm," pp. 282–286, 2019.

[16] M. Naz, Z. Anwaar, and W. H. Butt, "Automated white box test case generation for statement coverage using u-nsga-iii," in *2023 17th International Conference on Open Source Systems and Technologies (ICOSST)*, pp. 1–6, IEEE, 2023.

[17] Q. Shao, "Automatic case generation of variation testing in navigation software based on the genetic algorithm," pp. 263–268, 2023.

[18] A. S. Verma, A. Choudhary, and S. Tiwari, "Automatic test case generation framework for changed code using modified aeo algorithm in regression testing," pp. 81–84, 2023.

[19] M. Ahmed, A. B. Nasser, and K. Z. Zamli, "Construction of prioritized t-way test suite using bi-objective dragonfly algorithm," *IEEE Access*, vol. 10, pp. 71683–71698, 2022.

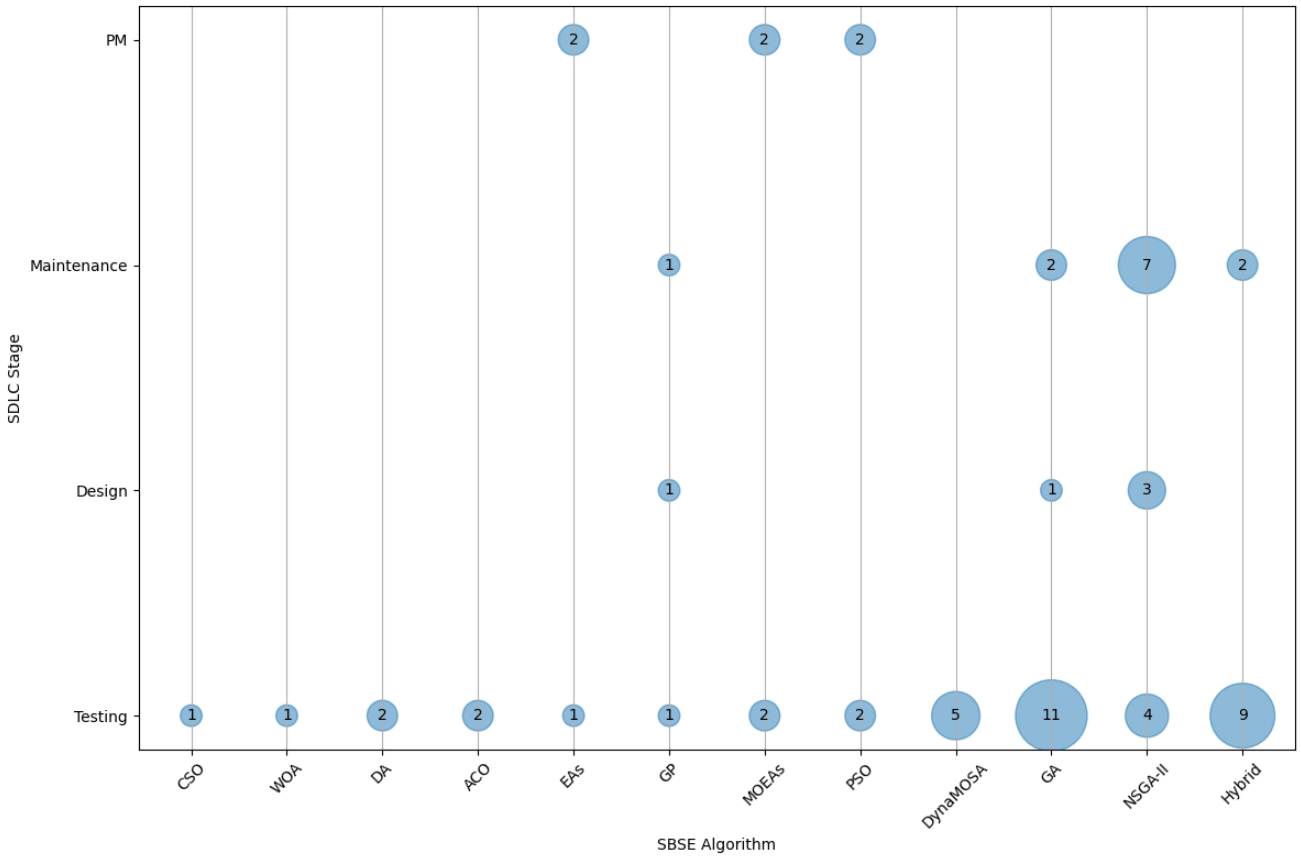

Fig. 11. Application of SBSE Algorithms Across SDLC Stages.

[20] H. B. Braiek and F. Khomh, "Deepevolution: A search-based testing approach for deep neural networks," pp. 454–458, 2019.

[21] H. N. N. Al-Sammarraie and D. N. Jawawi, "Multiple black hole inspired meta-heuristic searching optimization for combinatorial testing," *Ieee Access*, vol. 8, pp. 33406–33418, 2020.

[22] O. Al-Masri and W. A. Al-Sorori, "Object-oriented test case generation using teaching learning-based optimization (tlbo) algorithm," *IEEE Access*, vol. 10, pp. 110879–110888, 2022.

[23] A. Damia, M. Parvizimosaed, A. Bakhshai, and M. Salehi, "Optimized test data generation for path testing using improved combined fitness function with modified particle swarm optimization algorithm," in *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 412–416, IEEE, 2024.

[24] A. A. Muazu, A. S. Hashim, U. I. Audi, and U. D. Maiwada, "Refining a one-parameter-at-a-time approach using harmony search for optimizing test suite size in combinatorial t-way testing," *IEEE Access*, 2024.

[25] S. Potluri, J. Ravindra, G. B. Mohammad, and G. S. Sajja, "Optimized test coverage with hybrid particle swarm bee colony and firefly cuckoo search algorithms in model based software testing," in *2022 First International Conference on Artificial Intelligence Trends and Pattern Recognition (ICAITPR)*, pp. 1–9, 2022.

[26] G. Grano, C. Laaber, A. Panichella, and S. Panichella, "Testing with fewer resources: An adaptive approach to performance-aware test case generation," *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2332–2347, 2019.

[27] Z. J. Rashid and M. F. Adak, "Test data generation for dynamic unit test in java language using genetic algorithm," in *2021 6th International Conference on Computer Science and Engineering (UBMK)*, pp. 113–117, IEEE, 2021.

[28] S. D. Bejo, B. G. Assefa, and S. K. Mohapatra, "Backip: Mutation based test data generation using hybrid approach," in *2021 International Conference on Information and Communication Technology for Development for Africa (ICT4DA)*, pp. 178–183, IEEE, 2021.

[29] X. Dang, X. Yao, D. Gong, T. Tian, and B. Sun, "Multi-task optimization-based test data generation for mutation testing via relevance of mutant branch and input variable," *IEEE Access*, vol. 8, pp. 144401–144412, 2020.

[30] K. Serdyukov and T. Avdeenko, "Development and research of the test data generation approach modifications," in *2021 International Conference on Information Technology and Nanotechnology (ITNT)*, pp. 1–6, IEEE, 2021.

[31] M. R. H. Charmchi and B. R. Cami, "Paths-oriented test data generation using genetic algorithm," in *2021 12th International Conference on Information and Knowledge Technology (IKT)*, pp. 157–162, IEEE, 2021.

[32] Z. Cao, Y. Wang, P. Guo, and B. Tian, "Efsm test data generation based on fault propagation and multi-population genetic algorithm," in *2020 7th International Conference on Dependable Systems and Their Applications (DSA)*, pp. 240–245, IEEE, 2020.

[33] P. Chavan and P. Chavan, "An review on automated test data generation with java environment," in *2024 First International Conference on Pioneering Developments in Com-*

puter Science & Digital Technologies (IC2SDT), pp. 131–136, IEEE, 2024.

[34] F. Tang, "Design and java implementation of intelligent platform for english training based on intelligent test data generation algorithm," in *2022 3rd International Conference on Smart Electronics and Communication (ICOSEC)*, pp. 1641–1644, 2022.

[35] X. Dang, X. Yao, D. Gong, and T. Tian, "Efficiently generating test data to kill stubborn mutants by dynamically reducing the search domain," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 334–348, 2020.

[36] B. Arasteh, M. R. Sattari, and R. S. Kalan, *Fuzuli: Automatic Test Data Generation for Software Structural Testing using Grey Wolf Optimization Algorithm and Genetic Algorithm*. 2022.

[37] X. Yao, G. Zhang, F. Pan, D. Gong, and C. Wei, "Orderly generation of test data via sorting mutant branches based on their dominance degrees for weak mutation testing," *IEEE Transactions on Software Engineering*, vol. 48, no. 4, pp. 1169–1184, 2020.

[38] A. Ghaemi and B. Arasteh, "Sfla-based heuristic method to generate software structural test data," *Journal of software: Evolution and Process*, vol. 32, no. 1, p. e2228, 2020.

[39] X. Dang, W. Bao, Q. Qu, and D. Li, "Software testing combining the fusion of surrogate model and evolutionary algorithm," in *2023 International Conference on the Cognitive Computing and Complex Data (ICCD)*, pp. 311–316, IEEE, 2023.

[40] X. Yao, D. Gong, B. Li, X. Dang, and G. Zhang, "Testing method for software with randomness using genetic algorithm," *IEEE Access*, vol. 8, pp. 61999–62010, 2020.

[41] Y. Zhao, Y. Yang, Y. Zhou, and Z. Ding, "Depicter: a design-principle guided and heuristic-rule constrained software refactoring approach," *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 698–715, 2022.

[42] J. Liu, W. Jin, J. Zhou, Q. Feng, M. Fan, H. Wang, and T. Liu, "3erefactor: Effective, efficient and executable refactoring recommendation for software architectural consistency," *IEEE Transactions on Software Engineering*, 2024.

[43] V. Alizadeh, M. Kessentini, M. W. Mkaouer, M. Ó. Cinnéide, A. Ouni, and Y. Cai, "An interactive and dynamic search-based approach to software refactoring recommendations," *IEEE Transactions on Software Engineering*, vol. 46, no. 9, pp. 932–961, 2018.

[44] T. Ferreira, J. Ivers, J. J. Yackley, M. Kessentini, I. Ozkaya, and K. Gaaloul, "Dependent or not: Detecting and understanding collections of refactorings," *IEEE Transactions on Software Engineering*, vol. 49, no. 6, pp. 3344–3358, 2023.

[45] S. Rebai, V. Alizadeh, M. Kessentini, H. Fehri, and R. Kazman, "Enabling decision and objective space exploration for interactive multi-objective refactoring," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1560–1578, 2020.

[46] F. Adler, G. Fraser, E. Gründinger, N. Körber, S. Labrenz, J. Lerchenberger, S. Lukasczyk, and S. Schweikl, "Improving readability of scratch programs with search-based refactoring," pp. 120–130, 2021.

[47] V. Alizadeh, H. Fehri, and M. Kessentini, "Less is more: From multi-objective to mono-objective refactoring via developer's knowledge extraction," pp. 181–192, 2019.

[48] C. Abid, M. Kessentini, V. Alizadeh, M. Dhaouadi, and R. Kazman, "How does refactoring impact security when improving quality? a security-aware refactoring approach," *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 864–878, 2020.

[49] J. Zhang, X. Shen, and C. Yao, "Evolutionary algorithm for software project scheduling considering team relationships," *IEEE Access*, vol. 11, pp. 43690–43706, 2023.

[50] T. N. Bao, Q.-T. Huynh, X.-T. Nguyen, G. N. Nguyen, and D.-N. Le, "A novel particle swarm optimization approach to support decision-making in the multi-round of an auction by game theory," *International Journal of Computational Intelligence Systems*, vol. 13, no. 1, pp. 1447–1463, 2020.

[51] N. Nigar, M. K. Shahzad, S. Islam, O. Oki, and J. M. Lukose, "A novel multi-objective evolutionary algorithm to address turnover in the software project scheduling problem based on best fit skills criterion," *IEEE Access*, vol. 11, pp. 89742–89756, 2023.

[52] N. Nigar, M. K. Shahzad, S. Islam, S. Kumar, and A. Jaleel, "Modeling human resource experience evolution for multiobjective project scheduling in large scale software projects," *IEEE Access*, vol. 10, pp. 44677–44690, 2022.

[53] N. Nigar, M. K. Shahzad, S. Islam, O. Oki, and J. M. Lukose, "Multi-objective dynamic software project scheduling: A novel approach to handle employee's addition," *IEEE Access*, vol. 11, pp. 39792–39806, 2023.

[54] S. Akbar, M. Zubair, R. Khan, U. U. Akbar, R. Ullah, and Z. Zheng, "Weighted multi-skill resource constrained project scheduling: A greedy and parallel scheduling approach," *IEEE Access*, 2024.

[55] S. Fan, N. Yao, L. Wan, B. Ma, and Y. Zhang, "An evolutionary generation method of test data for multiple paths based on coverage balance," *IEEE Access*, vol. 9, pp. 86759–86772, 2021.

[56] S. D. Semujju, H. Huang, F. Liu, Y. Xiang, and Z. Hao, "Search-based software test data generation for path coverage based on a feedback-directed mechanism," *Complex System Modeling and Simulation*, vol. 3, no. 1, pp. 12–31, 2023.

[57] J. Goschen, A. S. Bosman, and S. Gruner, "Genetic microprograms for automated software testing with large path coverage," pp. 1–8, 2022.

[58] S. M. Al Khatib, "Optimization of path selection and code-coverage in regression testing using dragonfly algorithm," in *2021 International Conference on Information Technology (ICIT)*, pp. 919–923, IEEE, 2021.

[59] A. Perera, A. Aleti, M. Böhme, and B. Turhan, "Defect prediction guided search-based software testing," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 448–460, 2020.

[60] A. Perera, "Using defect prediction to improve the bug detection capability of search-based software testing," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1170–1174, 2020.

[61] J. Wang, S. Wang, J. Chen, T. Menzies, Q. Cui, M. Xie, and Q. Wang, "Characterizing crowds to better optimize worker recommendation in crowdsourced testing," *IEEE Transactions on Software Engineering*, vol. 47, no. 6, pp. 1259–1276, 2019.

[62] R. Casamayor, C. Cetina, O. Pastor, and F. Pérez, "Studying the influence and distribution of the human effort in a hybrid fitness function for search-based model-driven engineering," *IEEE Transactions on Software Engineering*, vol. 49, no. 12, pp. 5189–5202, 2023.

[63] A. H. F. Tabrizi and H. Izadkhah, *Software modularization by combining genetic and hierarchical algorithms*. 2019.

[64] M. Tajgardan, H. Izadkhah, and S. Lotfi, "A reinforcement learning-based iterated local search for software modularization," in *2022 8th Iranian conference on signal processing and intelligent systems (ICSPIS)*, pp. 1–6, IEEE, 2022.

[65] C. Schröder, A. van der Feltz, A. Panichella, and M. Aniche, "Search-based software re-modularization: a case study at adyen," pp. 81–90, 2021.

[66] M. Fathi and S. Khoshnevis, "Reusability metrics in search-based testing of software product lines: An experimentation," in *2021 26th International Computer Conference, Computer Society of Iran (CSICC)*, pp. 1–6, IEEE, 2021.

[67] A. A. Hassan, S. Abdullah, K. Z. Zamli, and R. Razali, "Combinatorial test suites generation strategy utilizing the whale optimization algorithm," *IEEE Access*, vol. 8, pp. 192288–192303, 2020.

[68] M. Panda, S. Dash, A. Nayyar, M. Bilal, and R. M. Mehmood, "Test suit generation for object oriented programs: A hybrid firefly and differential evolution approach," *IEEE Access*, vol. 8, pp. 179167–179188, 2020.

[69] Z. Su, G. Zhang, F. Yue, D. Zhan, M. Li, B. Li, and X. Yao, "Enhanced constraint handling for reliability-constrained multiobjective testing resource allocation," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 537–551, 2021.

[70] G. Zhang, L. Li, Z. Su, Z. Shao, M. Li, B. Li, and X. Yao, "New reliability-driven bounds for architecture-based multi-objective testing resource allocation," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2513–2529, 2022.

[71] N. Almarimi, A. Ouni, M. Chouchen, I. Saidani, and M. W. Mkaouer, "On the detection of community smells using genetic programming-based ensemble classifier chain," in *Proceedings of the 15th International Conference on Global Software Engineering*, pp. 43–54, 2020.

[72] G. Saranya, D. Mishra, V. Srikar, C. Abhilash, and S. Dooda, "Code smell detection using a weighted cockroach swarm optimization algorithm," in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 1–8, IEEE, 2023.

[73] B. Zhang, H. Zhang, J. Chen, D. Hao, and P. Moscato, "Automatic discovery and cleansing of numerical metamorphic relations," pp. 235–245, 2019.

[74] M. Bisi and V. Vishvkarma, "Software fault localization using jaya algorithm," in *2023 IEEE 20th India Council International Conference (INDICON)*, pp. 1076–1081, IEEE, 2023.

[75] J. Ivers, C. Seifried, and I. Ozkaya, "Untangling the knot: Enabling architecture evolution with search-based refactoring," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*, pp. 101–111, IEEE, 2022.

[76] L. C. Ochei, A. Petrovski, and J. M. Bass, "Optimal deployment of components of cloud-hosted application for guaranteeing multitenancy isolation," *Journal of cloud computing*, vol. 8, pp. 1–38, 2019.

[77] M. Einabadi and S. M. H. Hasheminejad, "A search-based method for optimizing software architecture reliability," in *2022 8th International Conference on Web Research (ICWR)*, pp. 47–54, IEEE, 2022.

[78] R. Ferdiana, A. E. Permanasari, *et al.*, "Complexity weights parameter optimization of use case points estimation using chaotic pso," in *2022 5th International Conference on Information and Communications Technology (ICOIACT)*, pp. 105–109, IEEE, 2022.

[79] D. V. Rodriguez and D. L. Carver, "Multi-objective information retrieval-based nsga-ii optimization for requirements traceability recovery," pp. 271–280, 2020.

[80] P. Zhu, Y. Li, T. Li, H. Ren, and X. Sun, "Advanced crowdsourced test report prioritization based on adaptive strategy," *IEEE Access*, vol. 10, pp. 53522–53532, 2022.

[81] B. Arasteh, F. S. Gharehchopogh, P. Gunes, F. Kiani, and M. Torkamanian-Afshar, "A novel metaheuristic based method for software mutation test using the discretized and modified forrest optimization algorithm," *Journal of Electronic Testing*, vol. 39, no. 3, pp. 347–370, 2023.

[82] K. M. Htay, R. R. Othman, A. Amir, H. L. Zakaria, and N. Ramli, "A pairwise t-way test suite generation strategy using gravitational search algorithm," pp. 7–12, 2021.

[83] M. Klima, M. Bures, and M. Blaha, "Ant colony optimization based algorithm for test path generation problem with negative constraints," in *2024 IEEE 24th International Conference on Software Quality, Reliability and Security (QRS)*, pp. 701–712, IEEE, 2024.

[84] J. Afonso and J. Campos, "Automatic generation of smell-free unit tests," pp. 9–16, 2023.

[85] M. Borg, R. B. Abdessalem, S. Nejati, F.-X. Jegeden, and D. Shin, "Digital twins are not monozygotic–cross-replicating adas testing in two industry-grade automotive simulators," pp. 383–393, 2021.

[86] S. Gerasimou, J. Cámara, R. Calinescu, N. Alasmari, F. Alhwikem, and X. Fang, "Evolutionary-guided synthesis of verified pareto-optimal mdp policies," pp. 842–853, 2021.

[87] D. Di Pompeo and M. Tucci, "Harnessing genetic improvement for sustainable software architectures," in *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*, pp. 248–249, IEEE, 2024.

[88] M. Akbari and H. Izadkhah, "Hybrid of genetic algorithm and krill herd for software clustering problem," pp. 565–570, 2019.

[89] F. H. Vera-Rivera, E. Puerto, H. Astudillo, and C. M. Gaona, "Microservices backlog–a genetic programming technique for identification and evaluation of microservices from user stories," *IEEE Access*, vol. 9, pp. 117178–117203, 2021.

[90] B. Zhang, G. Yi, Y. Wang, and Q. Fei, "Research on generation algorithm of soa-oriented integration test order," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 107–116, IEEE, 2021.

[91] F. H. Abba, K. Umar, U. A. Ibrahim, and A. I. Dalhatu, "Search-based prediction of software functional fault slip-through," in *2023 2nd International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS)*, pp. 1–7, IEEE, 2023.

[92] D. Sharma and S. Lohchab, "A search-based approach on metaheuristic algorithm for software modularization to optimize software modularity," in *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 1440–1450, IEEE, 2022.

[93] M. Soltani, A. Panichella, and A. Van Deursen, "Search-based crash reproduction and its impact on debugging," *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1294–1317, 2018.