e-Informatica

software engineering journal

2024 volume 18 issue 1

e-Informatica

e-Informatica

software engineering journal

2024 volume 18 issue 1

e-Informatica

Wrocław University
of Science and Technology

Faculty of Information and Communication Technology, Department of Applied Informatics
Wrocław University of Science and Technology,
50-370 Wrocław, Wybrzeże Wyspiańskiego 27, Poland

e-Informatica Software Engineering Journal
www.e-informatyka.pl, DOI: 10.37190/e-inf

Editorial Office Manager: Wojciech Thomas

Typeset by Wojciech Myszka with the LaTeX 2_ε Documentation Preparation System

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
www.oficyna.pwr.edu.pl;
e-mail: oficwyd@pwr.edu.pl; zamawianie.ksiazek@pwr.edu.pl

ISSN 1897-7979

Editorial Board

Contents

e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240101, DOI: 10.37190/e-Inf240101

A Multivocal Literature Review on Non-Technical Debt in Software Development: An Insight into Process, Social, People, Organizational, and Culture Debt

Hina Saeeda*, Muhammad Ovais Ahmad*, Tomas Gustavsson**
*Department of Computer Science, Karlstad University, Sweden
**Business School, Karlstad University, Sweden
Hina.saeeda@kau.se, Ovais.Ahmad@kau.se, Tomas.gustavsson@kau.se

Abstract

Software development encompasses various factors beyond technical considerations. Neglecting non-technical elements like individuals, processes, culture, and social and organizational aspects can lead to debt-like characteristics that demand attention. Therefore, we introduce the non-technical debt (NTD) concept to encompass and explore these aspects. This indicates the applicability of the debt analogy to non-technical facets of software development. Technical debt (TD) and NTD share similarities and often arise from risky decision-making processes, impacting both software development professionals and software quality. Overlooking either type of debt can lead to significant implications for software development success. The current study conducts a comprehensive multivocal literature review (MLR) to explore the most recent research on NTD, its causes, and potential mitigation strategies. For analysis, we carefully selected 40 primary studies among 110 records published until October 1, 2022. The study investigates the factors contributing to the accumulation of NTD in software development and proposes strategies to alleviate the adverse effects associated with it. This MLR offers a contemporary overview and identifies prospects for further investigation, making a valuable contribution to the field. The findings of this research highlight that NTD's impacts extend beyond monetary aspects, setting it apart from TD. Furthermore, the findings reveal that rectifying NTD is more challenging than addressing TD, and its consequences contribute to the accumulation of TD. To avert software project failures, a comprehensive approach that addresses NTD and TD concurrently is crucial. Effective communication and coordination play a vital role in mitigating NTD, and the study proposes utilizing the 3C model as a recommended framework to tackle NTD concerns.

Keywords: systematic reviews and mapping studies, software quality

1. Introduction

Software development is inherently a sociotechnical process, where the successful completion of software projects relies on the symbiotic relationship between technical capabilities and non-technical aspects of software development [1, 2]. This includes considering social aspects, as defects in software often arise from cognitive errors and miscommunication within and outside of organizations [3]. Such defective software leads to the accumulation

Submitted: 12 Dec. 2022; Revised: 23 Jun. 2023; Accepted: 19 Aug. 2023; Available online: 25 Sep. 2023

of technical debt and additional maintenance costs [4]. In software engineering, the term "technical debt" metaphorically describes the consequences of rushing software project development, resulting in defects and costly maintenance [5].

Over the past decade, both academia and industry have shown great interest in technical debt (TD) [3, 6], exploring various dimensions such as TD effort [7], TD tools [8], TD management strategies [5, 9], managing architectural TD [10], TD in Agile development [11], TD management elements [12], and TD prioritization [13]. Surprisingly, non-technical aspects also contribute to TD, as non-technical stakeholders play a role in driving projects to acquire TD [14]. Therefore in SD, the debt metaphor is used to describe issues prevalently – technical debt (i.e., code debt and code smells) and other debts (i.e., process debt, social debt, people debt, organisational, and cultural debt) [7, 10, 11, 13]. Software projects' success or failure combines technical and non-technical elements [11, 12]. Despite the significant attention given to TD in software [5–7, 10, 12, 14, 15], there remains a substantial research gap concerning the study of non-technical debt (NTD) [3, 7, 11, 13, 16–18]. NTD, such as process debt [7], people debt [16, 17], social debt [3, 11, 18–20], organizational debt [21], and cultural debt [13, 22] have not received sufficient consideration within the technical debt domain. A 2022 systematic mapping review identified a scarcity of scientific studies on NTD [1]. The review reported only 17 scientific studies on NTD and requested further empirical investigation as well as other forms of literature reviews to cover NTD in SD.

Therefore, this study aims to investigate NTD by conducting a multi-vocal literature review. According to Garousi et al. [23], when there is an absence of scientific evidence on any topic, it is recommended to conduct a multivocal literature review, as grey literature can provide valuable insights, perspectives, and empirical evidence that may not be available through traditional peer-reviewed sources. Including grey literature can lead to a more comprehensive understanding of the research topic. This multi-vocal literature review (MLR) provides a state-of-the-art of various NTD types, their causes, and mitigation strategies. The review extends and cross-validates the findings of a previously conducted systematic mapping review [1], enhancing the strength of the research outcomes by investigating similar research questions from different perspectives. By including scientific and grey literature, this review offers additional insights by capturing diverse perspectives and theoretical and practical insights. To achieve our study goals, we are investigating the research questions designed and reported in [1] that serve as the guided foundation for this MLR.

- RQ1 – What is the current state-of-the-art research on the different NTD types in software engineering?
- RQ2 – What are the reported causes of NTD in software engineering?
- RQ3 – What are the reported NTD mitigating strategies?
- RQ4 – What are the possible future directions for NTD in software development?

The present study is built upon previous research (doi.org/10.5220/0011772300003464) on the topic of NTD in SD. In this current version of our work, we aim to expand the scope of the prior investigation by offering a more comprehensive analysis. Specifically, we provide a detailed thematic division of the identified instances of NTD, including their underlying causes and potential solutions. Moreover, we conduct a thorough comparative analysis with a study referenced as [1]. The purpose of this comparison is to demonstrate how our research replicates, extends, and validates the existing body of work in this domain.

The rest of the study is structured as follows: The design and methodology of our investigation used in the research are explained in Section 2. Section 3 discusses the findings, and Section 4 is based on a discussion and conclusion. Section 5 examines threats to validity and how they were resolved. Finally, Section 6 represents future work.

2. Research method

This section outlines the Multivocal literature review (MLR) technique adopted in this study. We follow the established MLR guidelines and procedures proposed by Garousi et al. [23]. The complete systematic MLR process is illustrated in Figure 1, which consists of three phases: planning, conducting, and reporting. Each phase of the MLR study is discussed in detail in the rest of the section. Our MLR search was conducted on October 10, 2022, and analysis and reporting were completed by December 2022.

2.1. Planning the MLR

The primary purpose of conducting an MLR study is to focus on the "classification and thematic analysis of both scientific and grey literature on a software engineering topic" [23]. In this context, Garousi et al. [23] compared a systematic and a multivocal literature review study. A typical systematic literature review is motivated by a specific research topic that may be answered empirically. On the other hand, multivocal literature research examines a larger spectrum of software engineering challenges using grey and scientific literature. The following two processes (i.e., Motivation and Objectives and Research Questions) comprise the MLR planning phase, as depicted in Figure 1.

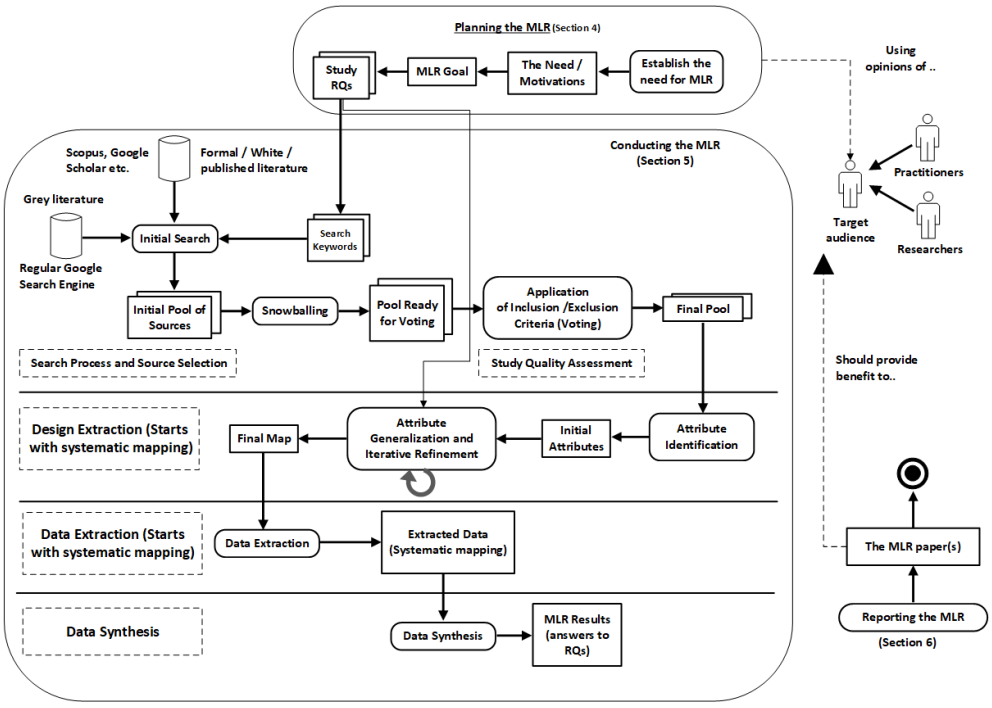


Figure 1. Complete MLR process

Motivation

MLR is useful in finding what is happening in an under-discovered phenomenon. MLR incorporates all accessible literature (including, but not limited to: blogs, white papers, articles, and academic literature) [23]. Therefore, MLR is vital for expanding research by

including non-scientific (grey) literature normally excluded in scientific studies. Academics and industry have been interested in TD for the last twenty years. Whereas the NTD is still in its infancy, it is necessary to investigate this niche and important research area further. In this case, an MLR study is motivated to learn about different types of NTD in software engineering and why they happen. What types of prevention or mitigation strategies may be utilized to prevent them?

The main objectives of this study are to understand: 1. What NTD is and how it can impact software engineering projects. 2. Identify the main types of NTD in software engineering projects. 3. Identify the main causes of NTD in software engineering projects. 4. Determine how to prevent or mitigate NTD in software engineering projects. 5. Determine the possibilities for future NTD research. These questions serve as the foundation for this study.

2.2. Conducting the MLR

At first, we conducted a pilot search on debt in software engineering using Google Scholar. The aim was to determine the existence of any secondary studies on the given topic. We conducted this search using the following string: ("process debt" OR "social debt" OR "people debt" OR "organizational debt" OR "culture debt") AND ("Software"). The search was conducted in October 2022, and Google Scholar yielded 3080 results. It was evident from the search that the current focus of research is predominantly on TD, whereas not a single review was found on NTD.

2.2.1. Search strategies and data sources

The designed search string defined the scope of our study. The designed string includes the search terms "population" and "intervention" based on (PICO) criteria suggested by Kitchenham et al. [24], where population refers to the application area, "software", and intervention represents NT'D types. Based on intervention, we selected five key terms for finalizing the search string (i.e., process debt, social debt, people debt, organizational debt, and cultural debt). Finally, the term software ensures we do not include research from other domains, like social sciences or economics. The finalized designed search string was ("process debt" OR "social debt" OR "people debt" OR "organizational debt" OR "culture debt") AND ("software").

The rationale for using the term "software" is that this study will cover studies that discuss software, software development, and systems. So, the search will include all documents with the word "software" in the title, abstract, and keyword. At the same time, the terms process debt, people debt, social debt, and organizational debt were used to include all NTD-associated sources. While we selected NTD (process, people, and social debts) from the existing systematic mapping review [1] on the topic and extended the scope of the study by adding two NTDs (cultural and organizational debts) as well. The overall motivation for selecting these five NTD types is based on the proposed "Hexagonal socio-technical systems framework" (adapted from Davis et al. 2014) [25] where they highlighted people, process, culture, and organization (technology and infrastructure) elements as the most critical sociotechnical aspects. As software development is an extensive interactive activity, we added the social debt [3] to cover the communication, collaboration, and cooperation challenges among people (directly or indirectly linked with the software development process). To ensure a broad overview of the topic, the selection criteria for including papers

in our study focus more on relevance. According to Lenarduzzi et al. [26], there are other types of debts, including service debt, suitability debt, and environmental sustainability debt, that can be studied under the umbrella of the NTD. However, in the current study, we are restricting our scope to only include the strongly linked social-technical elements of the software development process. Restricting the scope of our study has positive effects by enabling focused exploration, increased rigor, and efficient resource allocation. It helped us to concentrate on important research elements, leading to comprehensive findings. However, there are drawbacks to consider, including limited generalizability and potential oversight of relevant aspects.

The search string was designed to retrieve results from the Google search engine. We preferred the Google search engine as it is faster and a good source for collecting grey literature. We aimed to keep the search string simple to be as inclusive as possible with a new topic like non-technical debt in software engineering. Therefore, we did not restrict the search to particular years. We found 110 results with 11 Google pages, each with 11 links to further resources. Figure 2 shows the complete research conduction phases.

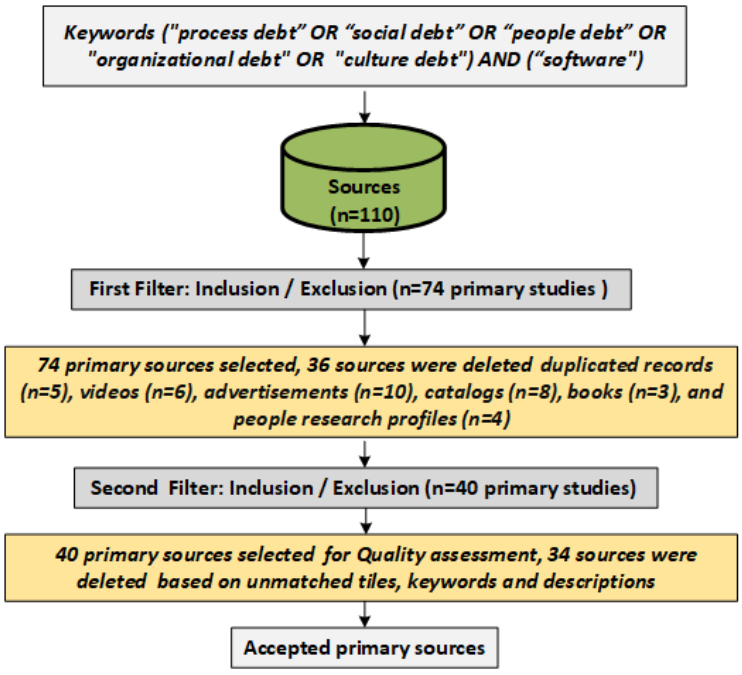


Figure 2. Research conduction process

2.2.2. Primary studies selection procedure and application of inclusion/exclusion criteria

In the first round, we excluded 36 records out of 110 based on the exclusion criteria given in Table 1. The detailed breakdown of excluded records is duplicated records ($n = 6$), videos ($n = 7$), advertisements ($n = 11$), catalogs ($n = 8$), and people research profiles ($n = 4$). After the first round, we obtained 74 records out of 110 and included them in subsequent steps.

Table 1. Inclusion and exclusion criteria

Inclusion	Exclusion
Sources contributing to the understanding of non-technical debt in software development, no matter to what extent the topic is discussed	Sources not written in English
Empirical and non-empirical sources, either qualitative or quantitative, analyzing any NTD in SD	Videos, advertisements, catalogs or keynotes, newspapers, duplicate sources
Blogs, white papers, and experiences report "people, process, social, cultural and organizational debt	Position and research profiles, non-software engineering domain sources

2.2.3. Quality assessment

To apply quality assessment criteria, the 40 primary studies [S1–S40] were divided into two categories: grey literature (GL) and scientific literature (SL). We adopted the 11-factor quality assessment criteria (Table 2) proposed by Dybå et al. [27] for scientific literature. At the same time, we adopted the quality assessment checklist of grey literature (Table 3) from Garousi et al. [23]. Each criterion was graded on a binary ("1" or "0") grade, in which "1" indicates "yes" to the question, while "0" means "no." Both checklist criteria measured the extent to which the quality of the 12 SL and 28 GL sources could be appropriately assessed. Two research separately consider the 40 primary sources. This technique helps to limit the degree of subjectivity and report the results more objectively. Researchers combined their results and solved a few conflicts in the discussion session.

Table 2. Quality assessment check list for SL

1.	Is the paper based on research (or is it merely a"lessons learned" report based on expert opinion)?
2.	Is there a clear statement of the aims of the research?
3.	Is there an adequate description of the context in which the research was carried out?
4.	Was the research design appropriate to address the aims of the research?
5.	Was the recruitment strategy appropriate to the aims of the research?
6.	Was there a control group with which to compare treatments?
7.	Was the data collected in a way that addressed the research issue?
8.	Was the data analysis sufficiently rigorous?
9.	Has the relationship between the researcher and participants been adequately considered?
10.	Is there a clear statement of findings?
11.	Is the study of value for research or practice?

For the scientific studies, based on the screening criterion, each of the 12 studies received a score of 1; each study offered a clear research objective and background for the investigation. However, one paper [7] lacked an adequate discussion of its research methodology and did not employ proper sampling. No relevant control group was found for comparing treatments in the primary studies. All primary publications adequately detailed "data collecting" and "data analysis", except [3]. While [7] lacks design and sampling phases. While "research findings" and "research value" criteria were applicable and fulfilled by all papers. Three papers [7, 16, 17] failed to discuss the researcher-participant connection explicitly. None of the papers received a complete score on the quality evaluation, but few publications received two or three negative responses. We divided grey literature into three tiers. We have 9 primary sources in 1st tier GL, which cover high outlet control/high credibility, including thesis, reports, and white papers. Two primary sources come under the umbrella of 2nd tier GL, which covers moderate outlet control/moderate credibility,

Table 3. Quality assessment check list for GL

1.	Is an individual author associated with a reputable organization?
2.	Has the author published other work in the field?
3.	Does the author have expertise in the area? (e.g., job title principal software engineer)
4.	Does the source have a clearly stated aim?
5.	Does the source have a stated methodology?
6.	Do authoritative, documented references support the source?
7.	Does the work cover a specific question?
8.	Does the work refer to a particular population?
9.	Is the work balanced in a presentation?
10.	Is the statement in the sources as objective as possible?
11.	Do the data support the conclusions?
12.	Does the item have a clearly stated date?
13.	Does it enrich or add something unique to the research?
14.	Does it strengthen or refute a current position
15.	16. 1st tier GL: High outlet control/High credibility: thesis, reports, white papers
17.	2nd tier GL: Moderate outlet control/Moderate credibility: Q/A sites, Wiki articles, workshop
18.	3rd tier GL: Low outlet control/Low credibility: Blog posts

including Q/A sites, Wiki articles, and workshops. There is 17 3rd-tier GL that cover low outlet control/low credibility. None of the GL sources received a complete score on the quality evaluation but reached the minimum threshold, which indicates credible sources as a whole. All the grey literature sources clearly stated their goal. Web blogs lack a methodology section and documented references, whereas thesis and seminar reports have written methodology sections. All sources provide information on specific NTD issues, cover the software development population, and are balanced in the overall presentation.

2.2.4. Data extraction and analysis

After completion of the quality analysis, data extraction was performed in MS Excel sheets based on the primary sources types, year of publication, NTD types found, NTD causes, and mitigation strategies. After the data extraction, data analysis was done using thematic analysis techniques [28]. The thematic analysis yielded primarily five themes, each highlighting NTD types. Further codes (discussed in the result section) were created against each NTD type, cause, and mitigation strategy.

3. Results

Our MLR study presents the results from analyzing the 40 primary sources [S1–S40] (see Appendix A). The presented results begin with demographic information, i.e., (i) type of literature, (ii) type of sources, and (iii) publication by year, and then proceed to a detailed assessment based on the thematic analysis.

3.1. Demographics

Figure 3 shows the two broader categories of our primary sources, GL ($n = 28$) and SL ($n = 12$). Figure 4 further shows the detailed breakdown of these two categories. The highest number of resources are cited from web blogs ($n = 17$), and the second highest number of resources are found in journals ($n = 6$), conferences ($n = 6$), and theses ($n = 6$). Also, seminar reports ($n = 3$) and two workshops are reported on the topic. This clearly

shows high practitioners' interest in the topic under study. Figure 5 shows that before 2018 fewer relevant studies were captured, whereas active research efforts have been evident since 2019.

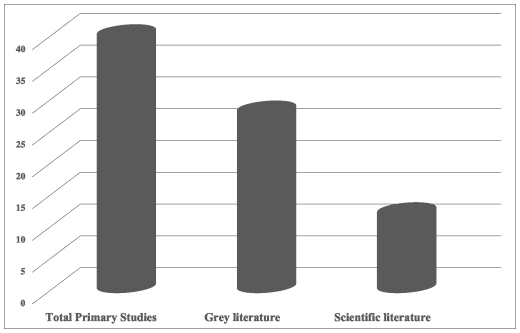


Figure 3. Type of literature

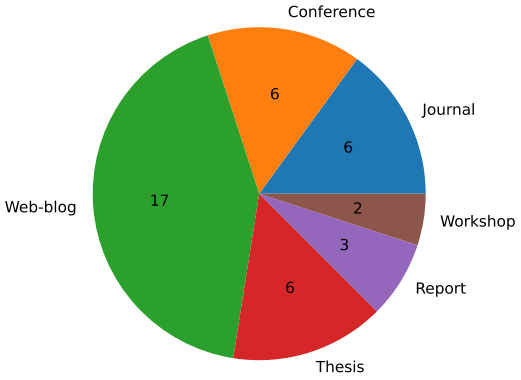


Figure 4. Types of primary sources

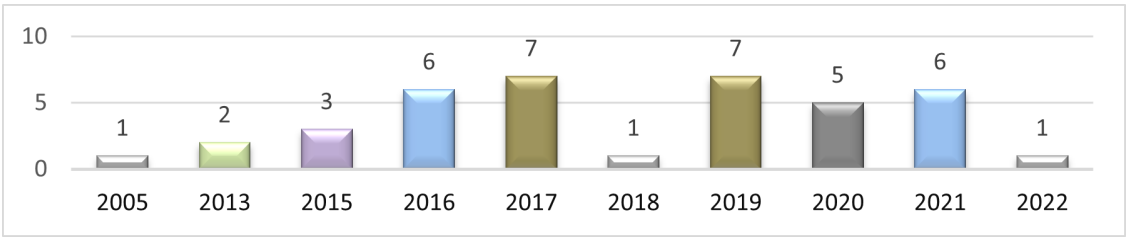


Figure 5. Publication's trend

3.2. NTD state of the art

In this section, we discuss five different types of NTD (i.e., people, process, social, cultural, and organization) with the help of relevant examples. This section answers RQ1 – What is the current state-of-the-art research on the different NTD types in software engineering?

3.2.1. Process debt

"Process debt refers to issues that, if present in software development, might delay or impede specific development operations" [29]. A software process is represented by a series of work phases applied to the design and development of a software product [30]. A process that operated effectively and efficiently a year ago may not be as productive as before, as changes in expectations, people, resources, and tools necessitate the modification of processes to achieve optimal performance. During these changes, the number of process bottlenecks, wasteful stages, and superfluous procedures add to process debt [31]. It occurs when a process is poorly understood and controlled [2]. Likewise, process debts are related to the creation of technical debt [7, 32]. Martini et al. [33] researched process debt in fine detail and revealed a number of its causes and mitigation strategies. An example of process debt is when teams hold stand-up meetings to report status so leaders know what is happening [16]. Team members can quickly show their leader the project's status. The debate focuses on recording rather than distributing information and addressing team interdependencies, causing process debt [16].

3.2.2. People debt

"People debt refers to people's problems in a software company that can slow or impede developmental operations" [5]. Human (people) factors in SD can be investigated psychologically, cognitively, managerially, and technically. These factors have organizational, interpersonal, and individual impacts. When building systems, enterprises accumulate people's debt by ignoring their people's needs. Unresolved people's issues [34], such as two people who had to be put on different teams because they could not work with each other, the person promoted whose head's only just been above water for the last six months is the typical causes of people debt [35–37]. It refers to all the ideas, goals, and objectives a corporation had for employees but abandoned. It also refers to missed opportunities to improve employee lives and jobs. One common example of people debt is expertise concentrated in too few people due to delayed training and/or hiring [2, 12, 16, 20]. Nokia's sharp fall, unveiled by an INSEAD study [38], resulted from a toxic culture of silence that the employees were experiencing that caused them to be in denial about the progress of their competition. Boeing's catastrophic failure resulted from an environment of fear, where engineers were unwilling to discuss problems and failures [34]. People's debt is also directly linked to the creation of TD [39].

3.2.3. Culture debt

"Cultural debt is making a technical decision that borrows against the organization's culture. Such decisions can introduce team divisions, deteriorate communication or even weaken leadership effectiveness" [13, 22]. Corroding culture affects morale and alienates partners, customers, and employees. The biggest threat to a company's ability to seize an opportunity isn't the wrong people, product, pricing, competition, or market forces; it's Culture Debt [35, 40]. Leaders often say, "Their employees are their greatest asset." If the culture is correct, if it is shattered and you hire out of step with it, everything will fail. These "biggest assets" will make unwise decisions and fail. The right people in the right atmosphere will always win [21, 40]. Uber had cultural debt as before 2017, the company's services, such as ordering a car, tracking a vehicle, and driver software, were packaged into one software. When part of the software crashed, the whole system went down [41]. The company switched software to recover swiftly and stabilize the business. The corporation bought 30 000 decentralized programs, which caused the technical debt. To cover the technical debt, they employed inexperienced workers without much training. The company's cultural debt produced decentralized communication, inadequate leadership, and disorganization.

3.2.4. Social debt

"Social debt is a cumulative and increasing cost in the current state of things, connected to invisible and negative effects within a development community" [16]. Tamburri et al. [3, 11, 18, 20] investigate social aspects of debt under the term "social debt." It is associated with unforeseen project costs, similar to technical debt, as it depicts the cost accumulation of software projects due to community causes, i.e., suboptimal working environments and conditions [42, 43]. Community causes contribute to the accumulation of social debt, which impacts the people who work on software development tasks and the quality of the software they produce [3, 27]. This chain of events can potentially jeopardize the business

continuity of software development organizations. An example of social debt is a lack of effective communication between different parts of the organization (for example, between development and operations teams). Another example has an architecture team that is disconnected from the SD team and, therefore, might suggest architectural solutions that are not realistic, as they do not take into consideration the details and the requirements elicited during the implementation phase [3, 36].

3.2.5. Organisational debt

Organizational debt is "the accumulation of changes that leaders should have made but didn't" [36]. It turns out that organizational debt can kill the company faster than other debts [42]. Organizational debt is all the people/culture compromises made to "just get it done" in the early stages of a startup. Growing companies must understand how to recognize and "refactor" organizational debt [27, 43, 44]. Organizational change requires individuals to change, whether introducing a new tool, restructuring business processes, or even larger transformation. These changes add to organizational debt as well [44]. The common organizational debt example is doing what's quick and convenient. It is understood that it does not scale and is not the ultimate solution, but it works now [5]. Further organizational debt example is obsolete processes. The company implemented a solution that worked at the time but now has better, more efficient options, but systems and processes rely on the old ways. Organizational debt is also directly linked to the generation of TD as well [34, 38, 45].

3.3. NTD accumulation causes

This section presents different identified common causes for accumulating the NTD. This section answers RQ2 – What are the reported causes of NTD in software engineering?

3.3.1. Process debt causes

Our study identified 14 process debt causes that are divided into three themes named process divergence, organizational and external dependencies (see Table 4). Among the identified causes, six were found to be discussed in both GL and SL, which include inefficient processes, outdated processes, sub-optimal processes, power distance, shortcuts, and quick fix norms prevalent in software companies. On the other hand, the GL introduced three new causes: lack of software culture, external trends, and technology and tools.

Process divergence. According to Martini et al. [7, 33], process designers develop defective processes, process executors deviate from well-designed processes, and infrastructure flaws can cause problems in process implementations. Process divergence is hard and can lead to inadequate planning, prioritization, and incompetency. Further causes of process debt are obsolete and suboptimal processes and lack of follow-up assessment, design, management, and execution [6, 7]. When process experts optimize processes, they add to process debt by missing some important steps. Unaccustomed staff causes confusion and process debt, too [7]. Inefficient processes include wasted time, customer delays, approval waits, batching delays, redundant steps, effort duplication, errors, and rework [15, 46, 48]. Obsolete or old processes include using outdated or time-consuming manual processes when a simple technological replacement could save time and costs [46, 48]. Inadequate defect analysis, documentation, or test case management causes process debt. No one uses more efficient

Table 4. Process debt causes

Process debt themes	Causes	GL ref.	SL ref.
Process divergence	Process incompetence		[33]
	Inefficient processes	[46]	[15]
	Outdate process	[29]	[47]
	Changing process	[48]	[20]
	Sub-optimal processes	[6]	[7]
	Lack of follow-up assessment		
	Lack of prioritization		
	Cost of process changes		
Organizational challenges	Value neglecting		
	Lack of software culture	[12]	
	Power distance	[49]	[50]
	Shortcuts and quick fix norms	[45]	[48]
External dependencies	Technology and tools		[7]
	External trends		

methods when shortcuts and quick solutions are the norms. This diminishes productivity when employees repeat the same "shortcut" when a permanent, more efficient approach might have been established from the start. Changing a software process changes activity, artifact, and role. One element's change may affect others due to interdependencies. Changing an activity (e.g., adopting agile) can influence final production. Unanalyzed changes in the processes can impair development [7, 47]. The value a process brings to stakeholders and the organization is sometimes unclear. Stakeholders disregard the process when such messages aren't properly communicated, causing process debt [6, 48]. Existing processes are hampered by different enterprises, units, teams, domains, and events. If these conditions and circumstances aren't considered, process debt and costly effects arise [7, 47]. Understanding contexts and scenarios in process design are vital; for example, when processes are built just for the software development team and neglect partnering hardware teams, it causes misunderstandings and generates process debt.

Organisational challenges. Power distance, lacking software culture, and neglecting values add to the accumulation of process debt. Power distance refers to the degree to which lower-ranking members of an organization "accept and expect the unequal distribution of authority". In the context of software development is the perceived distance between less powerful teammates and power-holder teammates, such as experienced teammates or decision-makers [12, 50]. The absence of a software culture in many organizations is a significant problem. This indicates that processes carefully crafted for use in other contexts may not be the most effective in all software development environments [7, 29]. Another issue is interacting with organizations with different cultures, interests, and power. It can lead to the negligence of important values; examples are processes not followed by open-source organizations developing a software system component used by the development team (e.g., the lack of a correct library versioning) or other stakeholders that may be interested in receiving data to compute analytics without knowing about the burden for the developers [49].

External dependencies. The influence of external trends, technology, and tools directly affects debt accumulation. How an organization adopts new processes is influenced by external trends such as greater global competition, changing demographics, changing customer concerns, and volatile stock markets. Process debt can occur when an organization adopts processes that are not fit for the organization based on these external trends. Having

the technology to support the processes, automate them, and make their steps easier is also essential. Additional process debt can be caused by a lack of technology and tools, such as when an outdated configuration management solution does not deliver quickly [7].

3.3.2. People debt causes

Our MLR identified 10 people's debt causes that are divided into three themes named behavioral, work, and 3C challenges (see Table 5). It is evident that people's debt is heavily investigated in the SL compared to GL, where seven cases are reported. On the other hand, GL offers two new causes named people´s poor behavior and remote work. Only two of the people's debts are identified in both SL and GL named race to ship as many features and poor inter-team coordination.

Table 5. People debt causes

People debt themes	Causes	GL ref.	SL ref.
Behavioural challenges	Frustrated and poor-performing teams		[12]
	Poor customer responsiveness		[12]
	People's poor behaviour	[36]	
Work challenges	Remote Work	[51]	
	Race to ship as many features	[51]	[16]
	Leaving people	[36, 51]	
3C challenges	Insufficient collaboration		[5]
	Insufficient communication		[5, 16]
	"Shortcuts" in communication		[5]
	Poor inter-team coordination	[38]	[5, 16]

Behavioural challenges. A team member's poor conduct negatively impacts the performance of other team members. Inversely, poor performance can also contribute to behavioral issues, particularly when team members become dissatisfied and angry about their poor performance or believe that an unfair standard has been established, which leads to poor productivity [12]. Low productivity has several detrimental repercussions on the workplace, including the financial impact on profitability and structural consequences on employee morale [12, 36]. Further poor behaviors of the people, such as excessive self-indulgence, a lack of self-control, exploiting others, and low motivation and effort, can be correlated with various antisocial, immoral, and imprudent behaviors that impede the software development process [36]. While poor customer responsiveness is based on the service provider's inability to provide in-time service, this is based on both the speed of interaction and the speed of the service fulfillment [12, 36].

Work challenges. In remote work without face-to-face connection, individuals miss a sense of shared purpose and are more indifferent to their employers. People's poor behaviors and attitudes toward work, not maintaining a positive attitude, shortcuts in communication, and remote working are also important reasons for people's debt [51]. Numerous disadvantages are linked with remote employment, as it directly impacts individuals' health. Loneliness is one of the primary obstacles that distant workers may need to overcome. When employees are not accustomed to working alone throughout the day, people could ask a coworker a brief inquiry or run into someone in the hallway to discuss casually while working in the office. Many employees miss a sense of shared purpose without face-to-face connection and are more indifferent to their employers. Another working challenge is the ambitious managers

who want to ship as many features as possible, which makes the development of the feature crawl over time. Also, when people leave, no one knows to whom the work should be transitioned. The combination of strict deadlines and excessive workloads causes software professionals to burn out and quit their careers prematurely [16]. Software engineers quit their jobs due to inadequate compensation in terms of money, technical difficulties, and growth opportunities. This has a detrimental effect on team performance; for example, the departure of team members is when no one understands who the task should be transferred to and who should fill the vacated position [36].

Communication, collaboration, and coordination (3C) challenges. During software development, "shortcuts in communication with people" refers to the incapacity of team members to interact effectively and appropriately [38]. This can occur when team members bypass proper communication channels, engage in extremely brief communication intervals, or avoid communication. Shortcuts in communication are taken to save time. Nevertheless, adopting communication shortcuts can impact multiple phases of the software development life cycle. For instance, omitting steps when speaking with the client to collect software requirements could result in a missed or insufficient requirements analysis. Failure to foster a collaborative environment at work results not only in a loss of benefits but also in a slew of disadvantages. The inability to create a team-friendly environment frequently results in an isolated and broken workflow, rarely leading to team efficiency or production [5]. When team members lack coordination, production can suffer, processes become more difficult, and work finishing can take longer [16].

3.3.3. Culture debt causes

Our MLR identified 7 cultural debt causes that are divided into two themes named organizational culture and management challenges (see Table 6). It is evident that cultural debt is heavily investigated in the GL and reported 6 causes. At the same time, unique causes identified in SL are named un-participatory culture. Here it is also worth mentioning that our MLR extended to cover cultural debt, which was not included in a recent NTD review conducted by Ahmad and Gustavsson [1]. This expansion is motivated by recognizing that a positive and productive work environment is essential for maximizing the effectiveness of processes and people involved in SD.

Table 6. Cultural debt causes

Culture debt themes	Causes	GL ref.	SL ref.
Org-culture challenges	Un-mindfulness in adopting culture	[52]	
	Un-participatory culture		[47]
	Individualistic culture	[22, 53]	
	Weak organisational culture	[13, 21]	
Management challenges	Managers lacking understanding of culture	[13]	
	Underinvestment in core HR functions	[13]	
	Hiring wrong people	[13, 54]	

Organisational culture challenges. Culture debt results from improperly implementing policies and procedures. Deferred investments because of budget uncertainty, inadequate governance, organizational restructuring, and the need to act rapidly to meet emergent threats are common cultural debt drivers in organizations [13, 53]. Unparticipatory culture oppressed active decision-making and goal-setting. Unparticipative environments make

employees feel less ownership over their work. They're more likely to ignore a problem or opportunity than assume someone else's responsibility. Individualistic cultures stress the needs of the individual over the needs of the group. In this type of culture, people are seen as independent and autonomous [21, 22]. Social behavior tends to be dictated by the attitudes and preferences of individuals. However, people with strong individualistic values and beliefs within an individualistic culture would have smaller social support networks, lower emotional competence, lower intentions to seek help from various sources, and poorer mental health [13, 22]. Individualism has also drawbacks where employees become too self-reliant, and a lack of emphasis on cooperation and teamwork leads to inefficiency in production [22]. Another big challenge faced is weak culture. It refers to values and beliefs not strongly and widely shared within the organization [13]. This implies that individual members of the organization rely more on personal principles, norms, and values. Further, the poorly implemented solution causes organizational culture to poorly identify the actions required, schedule the actions to identify the resources required, put measures in place to counter adverse consequences, and review the plans, etc. [13, 53].

Management challenges. When management teams don't know what culture they are trying to build in their organizations, it leads to cultural debt. Decision-makers have a cumulative impact on organizational culture. Managers can not guide the employees on how the company functions and is seen as a whole when they lack a proper understanding of the culture [13, 53]. Having adequate investments for managerial tasks is very important. Investing in management is important for cultural strength. Fewer management investments are a major reason for cultural debts in the software industry [13, 21]. Lack of investment in management causes degradation in revenue, branding, and workplace environment [13, 22]. The wrong hiring made by managers can have a serious and long-term impact on an organization. If a wrong hire is made, it can cause disruption in the team, increase recruitment and training costs, and decrease morale and productivity. It can also lead to a lack of trust in the leadership and a decrease in the quality of work. Additionally, the wrong hire can result in bad decision-making and cause the organization to miss out on potential opportunities [13].

3.3.4. Social debt causes

In the social debt category, we identified 17 causes that are divided into three themes named social confines, community smells, and organizational challenges (see Table 7). SL literature reported the majority of these cases, whereas GL reported only two new causes of social debt, named social isolation and decision in-communicability.

Social confines. Too much time alone at work and fewer collaborative talks contribute to social isolation [42, 51]. This hampers collaboration, idea-sharing, and teamwork. Social pressure is when one person or group influences another; for example, argument, persuasion, conformity, and demands are some social pressure examples [42]. Two types of social pressures exist 1. Workplace peer pressure, such as comparisons and competition with peers and 2. Psychological pressure: pressure caused by own thinking, such as overthinking, etc. Social pressure encourages improved performance and perfection. Seeing others succeed motivates others to achieve well. But extra and consistent social pressure on teams can lead to low self-esteem, lack of confidence, confusion about one's place in a social group, etc. A social structure is a network of (social) relationships, habits, and ways of thinking among people working toward a common goal. It helps people with the same organizational aim to communicate information. Communication deteriorates, work outputs are delayed,

Table 7. Social debt causes

Social debt themes	Causes	GL ref.	SL ref.
Social confines	Social isolation	[52]	
	Social pressure on teams		[47]
	Poor social structures		[50]
Organisational challenges	Uncooperativeness of the development community		[20]
	Organizational barriers		
	Uninformed socio-technical decisions		
	Architectural changes		
	Decision incommunicability	[36]	
	Global distance		[5]
	Lack of proper communication in organizations		[16]
	Omissions in social interactions		[43]
Community smells	Power distance		[42]
	Priggish members		
	Prima donnas		
	Radio silence or bottleneck		
	Haring villainy		
	Solution defiance		

and profitability is damaged when a company's social structure fails. To run effectively, it is necessary to frequently examine the social and organizational structure to ensure it matches the business's needs [42, 55].

Organisational challenges. The development community's unwillingness to work with technical experts causes organizational problems. Software development professionals rarely get along with techies. Unaware developers can create a "us vs. them" situation [3]. They blame each other for problems [3]. Organizational barriers impede employee knowledge flow and can lead to commercial failure [16, 20]. Organizational barriers include rules, policies, hierarchical positions, facilities, and complex systems. Employees must send queries in the organization's preferred language, medium, and manner of communication. The policy describes how employees should behave and communicate to stay employed [16, 18]. Uninformed socio-technical choices include poorly communicated organizational decisions and misinterpreted team findings. These ill-informed assessments can affect an organization's social and technical interdependence and the development community. It might lead to a lack of shared attention on communication and collaboration in achieving technical performance and job quality [3, 42]. Poor organizational or sociocultural conditions prevent the development network from communicating directly with key stakeholders [43]. Incommunicability is linked to communication and affected by social and organizational issues (e.g., organizational filtering protocols or nondisclosure agreements). Incommunicability traits, including community and smells, that impose communication obstacles (e.g., corporate silos or limited software practitioner communication cause social debt [5]. Organizational architecture increases societal debt. Architecture decisions can be determined "by osmosis," using information from every communication link in the development and operations network [12]. Critical information loss is almost inevitable. In the ensuing communication chain, important information, logic, and needs can be lost [20]. Inadequate communication, omission of social connections, and other issues contribute to social debt [5, 16].

Community smells. Priggish members refer to pedant teammates demanding of others pointlessly precise conformity or exaggerated propriety, which frustrates teammates and affects the software development process [18, 42]. Prima donnas work in isolation and

don't welcome changes or support from teammates. The outcome prevents the organization from innovative solutions or processes and effective communication and collaboration [3, 42]. Another form is a lone wolf, where individuals work regardless of their peers due to poor communication. The negative results of such work are unsanctioned architecture decisions across the development process, code errors, and project delays [20, 42]. Radio silence or bottleneck can occur when tasks and communications are formally performed in a complex organization [42]. For example, a team member working as a unique information intermediary for different teams leads to communication overload and massive delays [42]. Sharing villainy is an environment where the goal of sharing reliable knowledge or information is challenging. When organizations cannot offer a decent working environment and lack encouraging knowledge sharing, the result is that the team finds it difficult to complete project activities [42]. Solution defiance can be called team conflicts and a lack of respecting others' opinions regarding a potential solution. Each organization has teams that might be more diverse in various ways. Teams conflict in decision-making meetings when a team or individual is too rigid in their technical expertise, organizational cultural beliefs, values, and norms [55].

3.3.5. Organisational debt causes

Organizational debt leads to TD because of deferred investments due to budget uncertainty, poor governance and architecture, and organizational restructuring. On the other side, TD can also cause organizational debt. Non-technical debt refers to the broader organizational inefficiencies and operational shortcomings that hinder the organization's performance. Here's how system challenges can be related to non-technical debt in the organization. The inability to integrate systems, poor maintenance practices, the inherent complexity of systems, and complex, difficult-to-operate systems are considered causes of organizational debt. These factors can contribute to inefficiencies, increased costs, and hindered productivity within an organization. These system challenges can directly contribute to organizational debt. For a clear division of organizational debt themes, we have divided organizational challenges into two main types, i.e., organisational structure and system challenges [36]. Details of the organisational debt causes are provided in Table 8.

Table 8. Organizational debt causes

Organizational debt themes	Causes	GL ref.	SL ref.
Organizational structure challenges	Sluggish or inflexible organizations	[52]	
	Bad organizational, architectural choices	[52]	
	Lack of skills to upgrade organizations	[44]	
	Extensive and flat organization	[36]	
	Internal politics	[36]	
	Uneven information sharing among teams	[2]	
	Lack of management commitment	[56]	
	Unclear changes	[57]	
	Un structured Information	[57]	
	Lack of healthy communication	[41]	
	Organizational culture hindering progress	[41]	
Organizational-systems challenges	Inability to integrate systems	[52]	
	Poor maintenance	[52]	
	The inherent complexity of the systems	[52]	
	Complex, difficult to operate systems		[2]

Organisational structure challenges. Small and inflexible organizations lack the knowledge and capacity to upgrade and modernize [52]. Poor maintenance, inadequate investment, system complexity, and changing mission requirements affect them [44]. Bad organizational and architectural choices lead to bad decisions, performance evaluations, and compensation structures [58]. A very large and flat organizational structure has obstacles like a lack of hierarchical or flat structures: motivational issues, blurred decision-making processes, a lack of knowledge of areas of responsibility, and inconsistent processes and procedures. Flat organizations eliminate expectations. Some workers depart because they can't advance [36]. Internal politics (office or workplace politics) is inevitable. It refers to persons competing for prestige or power in the workplace. Politics decreases individual and organizational output. Politics harms the workplace [2]. Team members use their available informational resources through information sharing. Information sharing promotes innovation, efficiency, and new ideas by reducing repetition. Everyone benefits when employees share their knowledge and generate searchable content. Uneven information exchange across teams can harm the organization by hiding information from management [56]. Lack of management commitment generates an inefficient organization that hinders information sharing and cooperation. Lack of software implementation and maintenance resources affects quality [57]. When people don't grasp why change is needed, anxiety, scepticism, and resistance rise, and most significant changes are justified by financial returns (e.g., acquisitions add revenues; cost reductions increase margins). However, the rationale for large-scale change must be clear and convincing for all important stakeholders [41]. Unstructured information is difficult for people and computers to interpret. Unstructured information often causes workarounds that modern businesses don't understand. A lack of communication can produce misunderstandings, missed opportunities, conflict, disinformation, and mistrust, making staff feel defeated. Poor organizational communication causes friction, frustration, and confusion, producing a stressful climate where individuals aren't driven to collaborate or be productive. Culture impacts people's performance. Organizational culture is seen as a technique to get things done or as typical organizational features that shape member behavior and improve (or hinder) strategic achievement and performance [52]. Ambiguity, poor communication, and inconsistency are common cultural challenges. These can create a hostile and unpleasant workplace, leading to harassment, bullying, and high turnover. Culture drives growth and performance. Unhealthy workplace culture impairs engagement, retention, and performance. It hinders business when procedures and processes are structured to fit a legacy technology's capabilities. If you lack current collaborative tools like video conferencing or group chat, you may choose a local team over one with the best skills.

Organisational system challenges. A lack of or poorly designed integration can cause duplicate data, sluggish order processing, fulfillment delays, disgruntled customers, and profit loss. A lack of system integration hinders system performance and treasury operations with human work [52]. System integration challenges are caused by insufficient expertise, required money or investments, resources, inadequate communication/planning, after-go-live maintenance, and sophisticated technical concerns. Lack of integration produces information silos that obscure company performance. Inefficiencies hinder decision-making and raise redundancies [2]. Poor maintenance means failing to keep organizations working. Routine maintenance is preventive, predictive, or scheduled [52]. Maintenance is key to quality assurance and a company's long-term profitability. Unmaintained resources can cause instability and slow production. Malfunctioning machinery or breakdowns can be expensive. A software system's complexity isn't an accident. This intrinsic complexity originates from four elements: the complexity of the problem area, the difficulty of managing the

development process, software flexibility, and discrete system behavior difficulties [52]. Software complexity makes development management more difficult. Complex systems have many interacting parts. Hence they lack predictable causation. These components might alter over time, generating unpredictability in relationships. This causes unforeseen difficulties, defects, security failures, or crashes that are hard to examine. Detailed system descriptions, risk evaluations, and demand specifications are often wrong for systems where unexpected events occur [2].

Hence, the inability to integrate systems, poor maintenance practices, the inherent complexity of systems, and complex, difficult-to-operate systems are recognized causes of organizational debt. These factors directly contribute to inefficiencies, increased costs, and hindered productivity within an organization, thus adding to the burden of organizational debt. The inability to integrate systems effectively results in data inconsistencies, manual workarounds, and limited information flow, leading to higher costs, reduced productivity, and an accumulation of organizational debt over time. Poor maintenance practices, such as neglecting software updates and security patches, lead to degraded system performance, increased downtime, and higher maintenance costs, all of which contribute to organizational debt. The inherent complexity of systems, particularly legacy systems, requires specialized knowledge, training, and support, adding overhead costs and creating difficulties in system configuration, customization, and troubleshooting. Similarly, complex and difficult-to-operate systems with poor user interfaces and convoluted workflows impede employees' ability to perform tasks efficiently, resulting in errors, reduced productivity, and frustration. The time spent navigating these complex systems and seeking workarounds adds to inefficiencies and organizational debt.

3.4. NTD mitigation strategies

In this section, we are examining mitigation approaches for NTDs from the current literature. We will discuss each type of NTD management approach separately. This section answers RQ3 – What are the reported NTD mitigating strategies?

3.4.1. Process debt mitigation

We identified 5 process debt mitigation strategies (see Table 9). Only one mitigation strategy was identified in GL named measurement of process, whereas the remaining 4 are from SL. It is important to note that SLR [1] aims to provide information on how to prevent process debt and address architecture issues, requirement mismatches, process divergence, and organizational challenges. The current MLR, on the other hand, provides more specific recommendations and considerations for mitigation strategies. It highlights the importance of process documentation, monitoring, automation, market adaptation, and process design in managing process debt. It emphasizes the need for organizational restructuring involving end users. The MLR also suggests following conceptual models and tracking process productivity. The MLR offers broader principles and concepts for effective process debt management. It highlights the importance of process productivity tracking and conceptual models, which are not explicitly mentioned in [1].

Effective process debt management strategies are strongly tied to improving software development processes, such as process documenting, process monitoring, regular auditing, early detection of risks, and measuring process appropriateness [5, 29]. Process automation can minimize process debt [33], and by embracing new and valuable technologies and

tools, process automation can lead to functional augmentation and virtualization, making the process more understandable [33]. It is also a good idea to employ new processes to avoid process debt if market trends and needs change. Following the agile approach, for example, meeting early time to market, accepting dynamic requirements changes, engaging end users throughout the development process, and so on, all while revamping and improving overall organizational structures, aid in avoiding process debt. The changes to the development process are linked to the restructuring of the entire organization [2]. These redesigns are generally associated with organizational transformation, necessitating large-scale, broad-scale modifications.

Table 9. Process debt mitigation strategies

Mitigation strategies	GL ref.	SL ref.
Measurement of process	[29]	[5]
Automation of process		[2]
Continuous process assessment		[33]
Conceptual model for understanding the process debt		
Adopting new process		

An example is an organization's transition from a traditional software process to an agile software development process [33]. It is critical to have someone in charge of managing processes and supervisors who appreciate the importance of processes. Its primary purpose should be a continual evaluation that aids in proper process monitoring. On the other hand, a process must be designed with a specific purpose and value rather than just as a mandatory management tool imposed by the business. A thorough study is essential before selecting a process [16]. Researchers recommend following conceptual models to avoid process divergence and maintain track of process productivity in software development projects to understand process debt better [33].

3.4.2. People debt mitigation strategies

We identified 5 people's debt mitigation strategies (see Table 10). In the GL, two new mitigation strategies were identified, named work clubs' ideas for people's psyche and well-being and adopting the tradition of handling people's debt. Both GL and SL are reporting on educating business people about engineering people's decisions. In contrast, the remaining two strategies named continued monitoring and communication and managing dependencies, are reported in SL sources. Here it is important to note that the current MLR, in comparison to [1], delves deeper into the interpersonal and social aspects of managing people's debt, highlighting the significance of collaboration, resource provision, appreciation, education, and social support in creating a positive and supportive work environment.

People's debt management is difficult and uncontrollable since it is linked to human behavior, psyche, and well-being. But the people's challenges can be handled by encouraging collaboration among team members [59]. This could include having regular meetings where each person can discuss their progress and areas of improvement. Encourage team members to work together to find solutions to problems and foster open communication among team members [2, 60]. Clear communication between stakeholders and developers makes gathering adequate and clear requirements easy. Open communication also leads to better collaboration and monitoring of interdependencies between teams [2]. This could include

having team members provide honest feedback to each other and discussing their thoughts on how to improve the development process. Ensure all team members are on the same page regarding the project's goals. Everyone should understand what needs to be done and when it needs to be done [60]. This can help prevent the team from getting bogged down in the details and conflicts. Ensure everyone has the necessary resources to do their tasks [34]. This could include ensuring everyone can access the right tools and technology and providing support and guidance when needed [2]. Appreciate each team member for their hard work and dedication. This could include recognizing individual contributions and celebrating team successes. This can help motivate people to continue to do their best [15, 42]. Educating people adequately is also compulsory to avoid people's debt, such as educating businessmen about engineering people's decisions and to lessen business people's pressure on engineers by educating and communicating to them the technical perspectives [2]. People's well-being is strongly connected to their social ties and support; long periods of isolation at work are linked to stress, depression, and low morale. To keep people out debt, joint work groups or venues are recommended for daily discussion and social well-being. It is also important to continuously identify, prioritize, understand, and handle people's debt in organizations. So, there is a need to adapt to the tradition of people's debt understanding and handling.

Table 10. People debt mitigation strategies

Mitigation strategies	GL ref.	SL ref.
Continued monitoring and communication		[59, 60]
Work clubs' idea for people psyche and wellbeing	[39]	
Adopting tradition of handling people debt	[34]	
Managing dependencies		[37]
Educating business people about engineering people decisions	[2]	[42]

3.4.3. Culture debt mitigation strategies

We identified 4 culture debt mitigation strategies (see Table 11). Both GL and SL are reporting on three common mitigation strategies: handling issues collaboratively, delivering accurate information for intended multicultural audiences, clear communication, and an orderly business environment. While GL identified an additional mitigation strategy, creating the right mindset in the company. All of these findings are new to the existing literature as cultural debt is out of scope in [1].

To effectively handle cultural debt, organizations should focus on creating the correct mindset among their members. This includes fostering a growth mindset that embraces challenges and establishes trust within the working relationships. It is important for team members to feel comfortable discussing ideas, disagreements, and solutions. Encouraging diversity of thought is crucial for promoting creative problem-solving and innovation within teams [12, 13]. Encouraging team members to share different perspectives and ideas to see a problem from all angles and establishing clear communication between team members is one of the most effective ways to address cultural differences [54]. Respecting each other's cultural backgrounds and values may include being mindful of language and communication styles and understanding different approaches to problem-solving and decision-making, specifically by solving difficulties collaboratively across teams [22, 57] and enabling knowledge exchange with multicultural audiences [53, 56]. Creating more

natural workplaces through clear communication and an organized workplace also aids in regulating cultural debt. Investing in management for a better working culture and emphasizing organizational transparency provides a good culture [31].

Table 11. Cultural debt mitigation strategies

Mitigation strategies	GL ref.	SL ref.
Creating right mind set in company	[12, 13]	
Handle issues collaboratively	[50]	[47]
Deliver accurate information for intended multicultural audience	[57]	[53, 56]
Clear communication, orderly business environment	[22]	[5]

3.4.4. Social debt mitigation strategies

We identified 8 social debt mitigation strategies (see Table 12). Both GL and SL are reporting on three common mitigation strategies: social network analysis for debt analysis, collaborative communication, and guidelines for managing team composition and improving the description of architectural decisions. SL separately reports metrics for software architecture communicability and frameworks to follow for social debt mitigation, including the CAFFEA framework, architectural tactics, DAHLIA, and socio-technical quality framework. GL additionally reports combined work environments (hybrid) for social debt mitigation.

The current MLR expands the scope on topics not covered in study [1]. It emphasizes the significance of organizational strategies, frameworks, models, and guidelines that are crucial for monitoring and mitigating social debt. It recognizes the value of collaborative work environments and social network analysis as integral components of these strategies. It highlights the importance of fostering open communication, promoting collaboration, respecting diverse opinions, and implementing effective conflict-resolution strategies as essential measures for managing social debt. Additionally, the MLR acknowledges the significance of employing specific tools to diagnose and manage social debt within development communities.

In line with these findings, several organizational strategies, frameworks, models, tools, and guidelines help monitor and mitigate social debt. Social debt mitigation strategies are linked to collaborative work environments [32, 51] and social network analysis [20, 50]. Honest and open team communication and intense collaboration [11, 43]. Software development teams can manage social problems by encouraging open communication and collaboration, respecting the opinions of others, and using effective conflict resolution strategies [18, 19].

Open communication allows teams to share ideas and identify potential problems, while collaboration encourages everyone to work together to find solutions. Respect for the opinions of others is essential for teams to progress without disagreements or misunderstandings [19, 43].

Finally, effective conflict resolution strategies, such as brainstorming or seeking outside help, allow teams to work through disagreements and ensure everyone is on the same page. Practitioners must have the tools to diagnose and manage social debt in their development communities [18]. Some of the tools reported to detect and manage social debt are GEEZMO which alarms managers and supervisors about circumstances affecting teammates' mood; CodeFace4Smell detects organizational silos, black cloud, Lone wolf, and

Table 12. Social debt mitigation strategies

Mitigation strategies	GL ref.	SL ref.
Frameworks: 1. CAFFEA framework 2. Architectural tactics 3. DAHLIA 4. Socio-technical quality framework		[43, 50]
Combined work environments (hybrid)	[32, 51]	
Metric for software architecture communicability		[11]
Social network analysis for debt analysis	[18]	[20]
Collaborative communication	[19]	[43]
Guidelines: 1. Managing team composition 2. Improving the description of architectural decisions	[18]	[42, 50]
Tools: 1. GEEZMO 2. CodeFace4Smell 3. YOSHI		
Models: 1. Statistical 2. Social networks		

Radio silence social debt causes; DAHLIA, with key aspects, includes decision popularity, decision awareness to investigate some of the reasons of social debt [42].

3.4.5. Organisational debt mitigation strategies

We identified five organizational debt mitigation strategies (see Table 13). Among the strategies identified, only one was reported in both the SL and GL studies, namely monitoring, communication, and documentation. The remaining four strategies were exclusively documented in the GL, including adhering to the organizational model, maintaining and revising organizational charts and cloud architecture, and enhancing organizational culture. Notably, our study, the current MLR, focuses specifically on organizational debt mitigation strategies, which were not encompassed in a recent review on NTD [1].

Table 13. Organisational debt mitigation strategies

Mitigation strategies	GL ref.	SL ref.
Monitoring, communication and documentation	[45]	[43]
Following organizational model	[53, 57]	
Maintaining and revising organizational charts	[36, 44]	
Cloud architecture	[57]	
Improving organizational culture	[41, 49]	

The mitigation strategies emphasize on the importance of monitoring decisions and changes that need to be identified, prioritized, measured, and monitored. Such a monitoring process facilitates faster decision-making and drives business improvement by expediting reporting. Real-time exception detection plays a crucial role in enabling organizations to respond promptly to emerging challenges and opportunities [57]. These findings highlight

the significance of robust document management and adherence to an organizational structure based on validated frameworks.

Effective communication within an organization is essential for building trust, fostering teamwork, improving relationships, enhancing problem-solving abilities, and resolving conflicts [53, 57]. Robust document management practices ensure that all individuals within the company, regardless of their department or team, understand the storage, review process, and up-to-date status of documents. This ensures clarity and alignment and, if necessary, enables timely actions to be taken. Adhering to an organizational structure based on validated frameworks and updating organizational charts can streamline processes, enhance decision-making, manage multiple locations, drive employee performance, and prioritize customer service and satisfaction [43]. Social network analysis tools offer effective means to forecast, manage, and address debt within organizations [20]. Automating debt identification and testing tools enable continuous monitoring of debt sources and provide opportunities to proactively overcome them. In the dynamic software business, organizational adaptability and flexibility are crucial for survival and success [18, 36, 44].

Further cloud-based services ensure the continuity of organizational processes, reduce costs and foster increased collaboration. Cloud-based services are scalable and provide automatic software updates. It is not only efficient but also beneficial to the environment, and it provides automatic software integration [57]. The organizational culture can be improved by creating and communicating meaningful values to employees, conducting proper selection procedures, improving orientation and on-boarding for teams, enabling and empowering employees in skills and decisions, engaging employees in training, coaching according to their needs and domains, and communicating effectively and efficiently within teams [41, 55, 61].

4. Future work

Our MLR proposes several potential future directions that can further advance understanding of NTD and develop effective mitigation strategies. This section outlines key areas for future research in the field of NTD and answers RQ4. First, it would be interesting to apply social capital theory and control theory that can provide deeper insights into how social environments and relationships influence the accumulation of social and people debt. By leveraging these theories, researchers can explore the types of resources available through social networks and how they can be utilized to reduce debt. Second, develop a comprehensive taxonomy for categorizing and identifying distinct NTD types. This taxonomy can serve as a foundation for classifying NTD and enable the development of specialized approaches to tackle the unique challenges posed by each type. Stakeholders can benefit from this taxonomy by better understanding NTD types, their effects, and effective mitigation strategies. Third, investigating the effects of different NTD types on TD accumulation is crucial for comprehending how NTD leads to the creation of TD. Empirical studies are needed to explore the relationship between NTD and TD, considering factors such as poor planning, rushed coding, inadequate testing, and lack of refactoring. This research can shed light on how various issues contribute to the development of TD. Fourth, it's worth exploring other types of debts that can be studied under the umbrella of NTD, such as service debt, sustainability debt, and environmental sustainability debt. Investigating service debt can provide insights into trade-offs related to service-oriented architectures, service-level agreements, and service dependencies. Understanding and managing service

debt can contribute to the development of strategies and practices that ensure reliable and efficient service delivery. Exploring sustainability debt could focus on the long-term sustainability and maintainability of software systems. It aims to identify approaches that design and develop more sustainable software systems, both in terms of their technical aspects and their impact on the environment and society. Investigating environmental sustainability debt encompasses the ecological impact of software development activities, including energy consumption, resource utilization, and carbon emissions. Research in this area can contribute to the development of eco-friendly software engineering practices, reducing the environmental footprint of software systems. By pursuing these future research directions, researchers and practitioners can advance the understanding of NTD, develop effective mitigation strategies, and promote more sustainable and efficient software development practices.

5. Implications

Our MLR not only demonstrates the scarcity of research efforts on NTD, but it also demonstrates the direct relationship of NTD to TD in software development projects. It directs researchers to investigate further the relationship of NTD to TD in terms of causes, mitigation techniques, and consequences. This study elucidates the dangers of ignoring NTD while studying and developing TD. The highest number of grey literature studies shows the practitioners' increasing interest in the topic. However, it also forecasts that practitioners lack a holistic view of the different types of NTD and struggle to find a correlation between them. Most practitioners are aware of the TD and NTD relationship to it. Still, they do not understand how different NTD are connected. Therefore our study implicates the need for further investigation of TD to NTD and NTD to NTD relationships by conducting more industrial case studies. Our study also emphasizes practitioners' education and training about NTD and coping strategies for handling NTD in software development. Thus, we urge the companies to participate in research projects in the future to target research goals regarding the prevention and mitigation of NTD that are relevant to the software industry.

6. Threats to validity

We are addressing threats to external validity, threats to construct validity, and threats to conclusion validity. A data collection process was designed to support data recording to minimize the construct validity threat. Two researchers were involved in the whole process, which helped to lessen this threat even more. Because our MLR primary studies were largely based on online sources (grey literature), their applicability to the broader area of practices and general disciplines of TD and NTD is limited. We tried to minimize external validity threats by following the guidelines proposed by [23]. Conclusion validity is related to researchers' bias or misinterpretation of data. This is a major risk and cannot be eliminated. However, we took several steps to minimize this threat, such as having two researchers involved in the analysis, which helps limit subjective opinions. Further, a full audit and trial of 40 sources were maintained, and conclusions were drawn collaboratively.

7. Discussion and conclusion

The topic of debt is relatively new compared with other domains, such as software quality and testing. However, despite significant studies published on the TD concept, NTD has remained less explored. Among the different types of debt proposed in work by Lenarduzzi et al. [5], social debt has been investigated by [3] and process debt by [33]. While in a recent review conducted by Ahmad and Gustavsson [1], they extended the scope of previous studies by including an investigation of people's debt as well. However, they identified a scarcity of scientific studies on NTD [1]. They reported only 17 scientific studies on NTD and requested further empirical investigation and other forms of literature reviews on the topic. This MLR investigated NTD to extend the recently conducted systematic mapping review [1]. To achieve our study goals, we are investigating the research questions designed and reported in [1] that serve as the guided foundation for this MLR.

While TD resides within the system codebase [61], NTD seems more pervasive and intertwined with people, organizations, their working processes, and cultural issues. Software development is a socio-technical phenomenon based on socio-technical decisions. A socio-technical decision generates technical and any or all NTD types (i.e., people, process, culture, social, and organizational). TD outcomes are measured in monetary values, while consequences can measure NTD outcomes. We intentionally used the word ‘‘consequence‘‘ rather than ‘‘value" for NTD as its measurement is beyond the scope of monetary values. Therefore, more research is needed to understand how the effects of NTD can be measured or quantified in software projects. The results show that NTD seems harder to fix than TD. NTD contributes to TD accumulation, and its effects are both short- and long-term. NTD and TD are strongly intertwined with human dimensions – software architectures and their impact on businesses and cultures [3, 33, 42]. This highlights the complexity of managing debt in software development and underscores the need for further exploration and understanding of NTD to effectively mitigate its impact. NTD reflects and weighs heavily on the human and social aspects since it is caused by factors such as cognitive distance (lack of or excessive communication), mismatched architecture, and cultural and organizational systems [42]. This signal that NTD must be dealt with alongside TD to avoid severe consequences of software project failure.

Non-technical debt (NTD) significantly influences the human and social aspects of software projects, stemming from factors such as cognitive distance, architectural discrepancies, and cultural and organizational systems [42]. This highlights the need to address NTD alongside technical debt (TD) to mitigate the potentially severe consequences of project failure. In comparison to the existing review [1], our study uncovered a range of significant causes for process debt. These causes encompass process divergence, inefficient and outdated processes, changing process requirements, sub-optimal practices, lack of follow-up assessment, prioritization issues, costs associated with process changes, neglect of value considerations, power distance, reliance on shortcuts and quick fixes, as well as the influence of technology, tools, and external trends. This expanded our understanding of process debt causes and extended the scope of knowledge in this domain beyond the technology-focused causes identified in the systematic mapping review [1].

Regarding process debt mitigation, the SLR aims to provide general insights into preventing process debt and addressing architecture-related issues, requirement mismatches, process divergence, and organizational challenges [1]. In contrast, our study offers more specific recommendations and considerations for effective mitigation strategies. It highlights the importance of process documentation, monitoring, automation, market adaptation, and

thoughtful process design in managing process debt. Additionally, our study emphasizes the need for organizational restructuring that involves end users and suggests following conceptual models and tracking process productivity. These broader principles and concepts for effective process debt management supplement the recommendations provided by the recent review [1], as they explicitly address aspects such as process productivity tracking and the use of conceptual models.

For the people debt, the leading causes found were directly associated with inefficient collaboration, insufficient communication [51], shortcuts in communication [5, 42], and lack of inter-team coordination [16]. Therefore, it is clear that many issues linked to the causes of people's debt are based on a lack of communication and coordination. Behavioral challenges associated with people's debt arise from poor conduct and performance, decreasing productivity and customer responsiveness. Work challenges emerge from remote work disadvantages, such as a lack of shared purpose, poor attitudes, and ambitious managers causing feature development delays. Communication, collaboration, and coordination challenges manifest as shortcuts, inadequate teamwork, and coordination issues, impeding workflow and prolonging work completion. These challenges collectively impact the software development process, team performance, and productivity, necessitating effective solutions and strategies to mitigate their negative effects. The current MLR extends the understanding of people's debt in software development by highlighting specific behavior, work, and communication challenges compared to the recent review [1], which focuses on identifying factors such as knowledge gaps, inadequate management, and morale issues contributing to people's debt. For the mitigation of people's debt challenges, the current MLR, in comparison to review [1], delves deeper into the interpersonal and social aspects of managing people's debt, highlighting the significance of collaboration, resource provision, appreciation, education, and social support in creating a positive and supportive work environment.

The MLR expands the scope on topics of social debt not covered in the recent review [1]. It emphasizes the significance of organizational strategies, frameworks, models, and guidelines that are crucial for monitoring and mitigating social debt. It recognizes the value of collaborative work environments and social network analysis as integral components of these strategies. It highlights the importance of fostering open communication, promoting collaboration, respecting diverse opinions, and implementing effective conflict-resolution strategies as essential measures for managing social debt. Additionally, the current study acknowledges the significance of employing specific tools to diagnose and manage social debt within development communities.

We noticed that cultural and organizational terms were interchangeably used in the development communities, even though these terms have different meanings and contexts. While they can also have the exact reasons and mitigation strategies in specific contexts, i.e., carelessness in adopting policies, practices, and culture can lead to cultural debt in software development communities. It is also linked to the different causes of organizational debt, i.e., sluggish or inflexible systems, aging or legacy systems, and bad architectural choices associated with carelessness in adopting culture. This also leads to insufficient skills to upgrade and modernize the systems and infrastructure. In return, it leads to an inability to integrate systems and upgrade given organizational setups [52]. While we also noticed that there are two kinds of cultural perspectives in software development communities. (I) Work culture and (II) The employee's cultural background.

There is also a relationship between cultural and organizational impacts on selecting sub-optimal processes, which leads to the process debt in the result [33]. Therefore, poor

cultural and organizational choices are directly proportional to the selection of inefficient software development processes. Here it's important to note that both organizational and cultural debt that our MLR includes in the scope for investigation is not included in a recent NTD review conducted by Ahmad and Gustavsson [1]. The same pattern is reported in triggering social debt, i.e., lack of suitable communication among important sides of the organization [16] and missing social connections or reduced communication. The same applies to organizational debt, as the main reason for organizational debt causes is associated with uneven information sharing among teams [2] and a lack of healthy communication [48]. The second most important pattern we found is the relation of organizational debt with culture and software process, as poor organizational cultures are linked with hindering software development progress [41, 48]. Intuitively, smells that exist in community members' interactions hinder communication. Finally, cooperation is compromised by the smells existing in communities' structures. Businesses with inefficient processes and outdated software are also linked with organizational debt [41, 57]. So the analysis shows that "pinpointing" and separating different types of debt, i.e., technical debt and non-technical debt in SD, is challenging as they are greatly interlinked. NTD contributes to TD, and both cause equal damage. We further noticed that "All NTD contributes to TD," highlighting the interrelated nature of the technical and non-technical debt. It is also evident From the literature [3, 7, 11, 12, 16, 18, 20, 33, 55, 55, 62] that one type of NTD causes another type of NTD, i.e., culture debt, and organization debt can lead to process debt [7, 16, 33], people debt can lead to organizational debt, and culture debt [2, 53], and social debt can lead to people debt [13]. While they may be distinct types of debt, they are not mutually exclusive. In fact, non-technical debt can contribute to technical debt, as seen in the example above. Non-technical debt can create constraints that limit the ability to address technical debt or cause technical debt to be incurred in the first place. There is a clear connection between NTD and large-scale agile development. The challenges regarding testing strategies, specifically integration, regression, and user acceptance testing found in large-scale agile development, are reported as "test debt". At the same time, sprint-related challenges in agile projects are categorized as non-technical debt as well [1].

Acknowledgment

This research was performed within the Non-Technical Debt in Large-Scale Agile Software Development (NODLA) Project, funded by the Knowledge Foundation, Sweden.

References

[1] M.O. Ahmad and T. Gustavsson, "The Pandora's Box of social, process, and people debts in software engineering," *Journal of Software: Evolution and Process*, 2022, p. e2516.
[2] J. Yli-Huumo, *The role of technical debt in software development*, Ph.D. Thesis, Lappeenranta University of Technology, 2017.
[3] D.A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "Social debt in software engineering: Insights from industry," *Journal of Internet Services and Applications*, Vol. 6, 2015, pp. 1–17.
[4] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, Vol. 101, 2015, pp. 193–220.
[5] A. Melo, R. Fagundes, V. Lenarduzzi, and W.B. Santos, "Identification and measurement of requirements technical debt in software development: A systematic literature review," *Journal of Systems and Software*, 2022, p. 111483.

[6] P. Kruchten, R.L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, Vol. 29, No. 6, 2012, pp. 18–21.
[7] A. Martini and J. Bosch, "The danger of architectural technical debt: Contagious debt and vicious circles," in *12th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2015, pp. 1–10.
[8] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering," in *Dagstuhl Reports*, Vol. 6, No. 4. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016.
[9] W. Cunningham, "The WyCash portfolio management system," *ACM SIGPLAN OOPS Messenger*, Vol. 4, No. 2, 1992, pp. 29–30.
[10] N. Rios, R.O. Spínola, M. Mendonça, and C. Seaman, "The most common causes and effects of technical debt: First results from a global family of industrial surveys," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.
[11] D.A. Tamburri, "Software architecture social debt: Managing the incommunicability factor," *IEEE Transactions on Computational Social Systems*, Vol. 6, No. 1, 2019, pp. 20–37.
[12] J. Yli-Huumo, A. Maglyas, and K. Smolander, "How do software development teams manage technical debt? – An empirical study," *Journal of Systems and Software*, Vol. 120, 2016, pp. 195–218.
[13] A. Chen, *Cultural Debt*, 2022. [Online]. https://www.careerfair.io/reviews/cultural-debt
[14] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, "An enterprise perspective on technical debt," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, 2011, pp. 35–38.
[15] J. Yli-Huumo, A. Maglyas, and K. Smolander, "The effects of software process evolution to technical debt – Perceptions from three large software projects," *Managing Software Process Evolution: Traditional, Agile and Beyond – How to Handle Process Change*, 2016, pp. 305–327.
[16] A. Martini, V. Stray, and N.B. Moe, "Technical, social and process debt in large-scale agile: An exploratory case-study," in *Agile Processes in Software Engineering and Extreme Programming – Workshops: XP Workshops*. Montréal, QC, Canada: Springer, 2019, pp. 112–119.
[17] Z. Li, P. Liang, and P. Avgeriou, "Architectural debt management in value-oriented architecting," in *Economics-Driven Software Architecture*. Elsevier, 2014, pp. 183–204.
[18] D. Tamburri, *From Technical to Social Debt: Analyzing Software Development Communities using social networks analysis*, 2015. [Online]. https://www.slideshare.net/DamianTamburri/from-technical-to-social-debt-analyzing-software-development-communities-using-socialnetworks-analysis
[19] T. Mejía, *Social Debt: the difficult commitment*, 1998. [Online]. https://www.socialwatch.org/book/export/html/10623
[20] D.A. Tamburri and E. Di Nitto, "When software architecture leads to social debt," in *12th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2015, pp. 61–64.
[21] C.B. Jaktman, "The influence of organizational factors on the success and quality of a product-line architecture," in *Proceedings 1998 Australian Software Engineering Conference*. IEEE, 1998, pp. 2–11.
[22] B. Sutton, *Overcoming Cultural and Technical Debt*, 2019. [Online]. https://sloanreview.mit.edu/audio/overcoming-cultural-and-technical-debt/
[23] V. Garousi, M. Felderer, and M.V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Information and Software Technology*, Vol. 106, 2019, pp. 101–121.
[24] B. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey et al., "Systematic literature reviews in software engineering – A systematic literature review," *Information and Software Technology*, Vol. 51, No. 1, 2009, pp. 7–15.
[25] M.C. Davis, R. Challenger, D.N. Jayewardene, and C.W. Clegg, "Advancing socio-technical systems thinking: A call for bravery," *Applied Ergonomics*, Vol. 45, No. 2, 2014, pp. 171–180.

[26] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F.A. Fontana, "A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools," *Journal of Systems and Software*, Vol. 171, 2021, p. 110827.
[27] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, Vol. 50, No. 9–10, 2008, pp. 833–859.
[28] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, Vol. 3, No. 2, 2006, pp. 77–101.
[29] L. McGuire, *What Is Process Debt, and Why Is It a Problem*, 2022. [Online]. https://www.formstack.com/blog/process-debt
[30] I. Sommerville, *Software Engineering*, 9th ed. Addison-Wesley, 2011.
[31] S.W. Wenger E, McDermott RA, *Cultivating Communities of Practice: a Guide to Managing Knowledge. Harvard Business School Publishing*, 2002. [Online]. https://hbswk.hbs.edu/archive/cultivating-communities-of-practice-a-guide-to-managing-knowledge-seven-principles-for-cultivating-communities-of-practice
[32] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *2009 20th International Symposium on Software Reliability Engineering*. IEEE, 2009, pp. 109–119.
[33] A. Martini, T. Besker, and J. Bosch, "Process debt: A first exploration," in *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2020, pp. 316–325.
[34] D. Blomstrom, *How to Recognise and Reduce HumanDebt*, 2022. [Online]. https://www.infoq.com/articles/human-debt
[35] P. Vinayak, *Everything you need to know about Cultural debt*, 2021. [Online]. https://e2ehiring.com/blogs/everything-you-need-to-know-about-cultural-debt
[36] M. Bellotti, *Hunting Tech Debt via Org Charts. Knowing where to look for problems*, 2021. [Online]. https://bellmar.medium.com/hunting-tech-debt-via-org-charts-92df0b253145
[37] L. Pirzadeh, "Human factors in software development: A systematic literature review," Master's thesis, 2010.
[38] INSEAD, "Company strategic planning/Interviews/INSEAD," 2015.
[39] G. Marlow, *People debt is like technical debt – eqsystems.io*, 2017. [Online]. https://eqsystems.io/2017/04/people-debt-like-technical-debt
[40] B. Coleman, *Culture Debt Is One of the Most Toxic Threats to Business, and Your Startup Is Probably Victim to It*, 2019. [Online]. https://www.inc.com/bernard-coleman/culture-debt-is-one-of-most-toxic-threats-to-business-your-startup-is-probably-victim-to-it.html
[41] M. Hosking, *Transformation troubles and non-technical debt*, 2017. [Online]. https://www.linkedin.com/pulse/transformation-troubles-non-technical-debt-matt-hosking/
[42] E.A.C. Espinosa, *Understanding Social Debt in Software Engineering*, Ph.D. dissertation, The University of Alabama, 2021.
[43] T. Dreesen, P. Hennel, C. Rosenkranz, and T. Kude, ""The second vice is lying, the first is running into debt." Antecedents and mitigating practices of social debt: An exploratory study in distributed software development teams," in *Proceedings of the 54th Hawaii International Conference on System Sciences*, 2021, p. 6826.
[44] J. Trouw, *Organisational debt an analogy*, 2021. [Online]. https://www.linkedin.com/pulse/organisational-debt-analogy-jaap-trouw
[45] S. Blank, "Organizational debt is like technical debt – but worse," 2015. [Online]. https://www.forbes.com/sites/steveblank/2015/05/18/organizational-debt-is-like-technical-debt-but-worse-2/?sh=6ea3ce447b35
[46] S. Priestnall, *What is Process Debt?*, 2020. [Online]. https://www.linkedin.com/pulse/what-process-debt-steve-priestnall
[47] J.A. Miko, *Collaboration strategies to reduce technical debt*, Ph.D. dissertation, Walden University, 2017.
[48] M. Eaden, *When Testers Deal With Process Debt: Ideas to Manage It and Get Back to Testing Faster*, 2017. [Online]. https://www.ministryoftesting.com/articles/8d79968d?s_id=15650023

[49] L.P. Gates, *Are We Creating Organizational Debt*, 2017. [Online]. https://insights.sei.cmu.edu/blog/are-we-creating-organizational-debt/
[50] R. Kazman, "Managing social debt in large software projects," in *IEEE/ACM 7th International Workshop on Software Engineering for Systems-of-Systems (SESoS) and 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (WDES)*. IEEE, 2019, p. 1.
[51] K. Ladewig, *The dark side of working from home*, 2019. [Online]. https://medium.com/swlh/social-debt-17bf03a269a
[52] S. Vinsennau, *Decouple to innovate how federal agencies can unlock IT value and agility by remediating technical debt*, 2016. [Online]. https://www.accenture.com/_acnmedia/PDF-85/Accenture-Decoupling-to-Innovate.pdf
[53] B. Falchuk, *What's the Greatest Threat to Your Organization? Culture Debt*, 2019. [Online]. https://bryanfalchuk.com/blog/culture-debt
[54] B. Coleman, *Culture Debt Is One of the Most Toxic Threats to Business, and Your Startup Is Probably Victim to It*, 2019. [Online]. https://www.inc.com/bernard-coleman/culture-debt-is-one-of-most-toxic-threats-to-business-your-startup-is-probably-victim-to-it.html
[55] F. Palomba, D.A. Tamburri, F.A. Fontana, R. Oliveto, A. Zaidman et al., "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE Transactions on Software Engineering*, Vol. 47, No. 1, 2018, pp. 108–129.
[56] D. O'Keeffe, "An empirical case study of technical debt management: A software services provider perspective," M.Sc. thesis, University of Dublin, 2017.
[57] A. Dignan, *How to Eliminate Organizational Debt – Building Strong Organizations*, 2017. [Online]. https://culturestars.com/how-to-eliminate-organizational-debt
[58] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," in *Proceedings of the 30th International Conference on Software Engineering*, 2008, pp. 521–530.
[59] J. Cusick and A. Prasad, "A practical management and engineering approach to offshore collaboration," *IEEE Software*, Vol. 23, No. 5, 2006, pp. 20–29.
[60] C.R. De Souza and D.F. Redmiles, "The awareness network, to whom should i display my actions? and, whose actions should i monitor?" *IEEE Transactions on Software Engineering*, Vol. 37, No. 3, 2011, pp. 325–340.
[61] K. Casey, *What causes technical debt – and how to minimize it*, 2020. [Online]. https://enterprisersproject.com/article/2020/6/technical-debt-what-causes
[62] L.M. Hilty and B. Aebischer, "ICT for sustainability: An emerging research field," *ICT Innovations for Sustainability*, 2015, pp. 3–36.

Appendix A. Primary studies

List of primary studies

[S1] J. Yli-Huumo, A. Maglyas, and K. Smolander, "How do software development teams manage technical debt?–an empirical study," *Journal of Systems and Software*, Vol. 120, 2016, pp. 195–218.
[S2] A. Martini and J. Bosch, "Revealing social debt with the caffea framework: An antidote to architectural debt," in *IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 179–181.
[S3] K. Ladewig, *The dark side of working from home | The Startup*, 2019. [Online]. https://medium.com/swlh/social-debt-17bf03a269a
[S4] S. Sachdev, *Cultural Debt*. [Online]. https://www.careerfair.io/reviews/cultural-debt
[S5] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering," in *Dagstuhl Reports*, Vol. 6, No. 4. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016.

[S6] A. Melo, R. Fagundes, V. Lenarduzzi, and W.B. Santos, "Identification and measurement of requirements technical debt in software development: A systematic literature review," *Journal of Systems and Software*, 2022, p. 111483.
[S7] D.A. Tamburri, "Software architecture social debt: Managing the incommunicability factor," *IEEE Transactions on Computational Social Systems*, Vol. 6, No. 1, 2019, pp. 20–37.
[S8] T. Mejía, *Social Debt the difficult commitment=* , 1998. [Online]. https://www.socialwatch.org/book/export/html/10623
[S9] S. Vinsennau, *Decouple To Innovate How Federal agencies can unlock IT value and agility by remediating technical debt*, 2016. [Online]. https://www.accenture.com/_acnmedia/PDF-85/Accenture-Decoupling-to-Innovate.pdf
[S10] G.S. Tonin, *Technical debt management in the context of agile methods in software development*, Ph.D. dissertation, University of Sao Paulo, 2018.
[S11] T. Besker, H. Ghanbari, A. Martini, and J. Bosch, "The influence of technical debt on software developer morale," *Journal of Systems and Software*, Vol. 167, 2020, p. 110586. [Online]. https://www.sciencedirect.com/science/article/pii/S0164121220300674
[S12] M. Bellotti, *Hunting Tech Debt via Org Charts. Knowing where to look for problems*, 2021. [Online]. https://bellmar.medium.com/hunting-tech-debt-via-org-charts-92df0b253145
[S13] J. Yli-Huumo, *The role of technical debt in software development*, Ph.D. dissertation, Lappeenranta University of Technology, 2017.
[S14] S. Priestnall, *What is Process Debt?*, 2020. [Online]. https://www.linkedin.com/pulse/what-process-debt-stevepriestnall
[S15] Z. Dargó, "Technical debt management: Definition of a technical debt reduction software engineering methodology for smes," Master's thesis, Aalto University, School of Science, 2019.
[S16] J.A. Miko, *Collaboration Strategies to Reduce Technical Debt*, Ph.D. dissertation, Walden University, College of Management and Technology, 2017.
[S17] B. Sutton and P. Michelman, *Overcoming Cultural and Technical Debt*, MITSloan Management Review, 2019. [Online]. https://sloanreview.mit.edu/audio/overcoming-cultural-and-technical-debt/
[S18] D.A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "What is social debt in software engineering?" in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2013, pp. 93–96.
[S19] D.A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "Social debt in software engineering: insights from industry," *Journal of Internet Services and Applications*, Vol. 6, 2015, pp. 1–17.
[S20] D. O'Keeffe, "An empirical case study of technical debt management: A software services provider perspective," Master's thesis, University of Dublin, 2017.
[S21] B. Coleman, *Culture Debt Is One of the Most Toxic Threats to Business, and Your Startup Is Probably Victim to It*, 2019. [Online]. https://www.inc.com/bernard-coleman/culture-debt-is-one-of-most-toxic-threats-to-business-your-startup-is-probably-victim-to-it.html
[S22] A. Dignan, *How to Eliminate Organizational Debt – Building Strong Organizations*, 2017. [Online]. https://culturestars.com/how-to-eliminate-organizational-debt
[S23] D.A. Tamburri and E. Di Nitto, "When software architecture leads to social debt," in *2015 12th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2015, pp. 61–64.
[S24] A. Martini, V. Stray, and N.B. Moe, "Technical-, social-and process debt in large-scale agile: an exploratory case-study," in *Agile Processes in Software Engineering and Extreme Programming–Workshops: XP 2019 Workshops, Montréal, QC, Canada, May 21–25, 2019, Proceedings 20*. Springer, 2019, pp. 112–119.
[S25] A. Martini, T. Besker, and J. Bosch, "Process debt: A first exploration," in *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2020, pp. 316–325.
[S26] S. Zimmeck, *Social Debt: Why Software Developers Should Think Beyond Tech*, 2019. [Online]. https://sebastianzimmeck.medium.com/social-debt-why-software-developers-should-think-beyond-tech-df665d8401a5

[S27] P. Phelan, *Is "Cultural Debt" hurting your organization's growth? (Part 1)*, 8W8 Global Business Builders. [Online]. https://www.8w8.com/is-cultural-debt-hurting-your-organizations-growth-part-1/

[S28] J. Trouw, *Organisational debt an analogy= https://www.linkedin.com/pulse/organisational debt analogy*, 2021. [Online]. https://www.linkedin.com/pulse/organisational-debt-analogy-jaap-trouw

[S29] D. Tamburri, *From Technical to Social Debt: Analyzing Software Development Communities using social networks analysis*, 2015. [Online]. https://www.slideshare.net/DamianTamburri/from-technical-to-social-debt-analyzing-software-development-communities-using-socialnetworks-analysis

[S30] J. Holvitie, D. Tamburri, A. Goldman, S. Fraser, W. Snipes et al., "Social debt in software engineering: Towards a crisper definition," Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Seminar 16162, 2016. [Online]. https://www.dagstuhl.de/16162

[S31] R. Kazman, "Managing social debt in large software projects," in *2019 IEEE/ACM 7th International Workshop on Software Engineering for Systems-of-Systems (SESoS) and 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (WDES)*. IEEE, 2019, pp. 1–1.

[S32] T. Dreesen, P. Hennel, C. Rosenkranz, and T. Kude, ""the second vice is lying, the first is running into debt." antecedents and mitigating practices of social debt: An exploratory study in distributed software development teams," in *Proceedings of the 54th Hawaii International Conference on System Sciences*, 2021, p. 6826.

[S33] F. Palomba, A. Serebrenik, and A. Zaidman, "Social debt analytics for improving the management of software evolution tasks," in *16th Edition of the BElgian-NEtherlands Software EVOLution Symposium (BENEVOL 2017)*. CEUR-WS.org, 2017, pp. 18–21.

[S34] M. Eaden, *When Testers Deal With Process Debt: Ideas to Manage It And Get Back To Testing Faster*, 2017. [Online]. https://www.ministryoftesting.com/articles/8d79968d?s_id=15650023

[S35] E.A. Caballero Espinosa, *Understanding Social Debt in Software Engineering*, Ph.D. dissertation, The University of Alabama, 2021.

[S36] W. Cunningham, "The wycash portfolio management system," *ACM SIGPLAN OOPS Messenger*, Vol. 4, No. 2, 1992, pp. 29–30.

[S37] I. Kavas, "Don't go back to the office without fixing your process debt," *Forbes*, 2021. [Online]. https://www.forbes.com/sites/forbestechcouncil/2021/01/04/dont-go-back-to-the-office-without-fixing-your-process-debt/?sh=1afbfe9b74a4

[S38] *Organisational Debt and Why It Makes Digital Transformation Hard*, CloudThing, 2022. [Online]. https://cloudthing.com/organisational-debt/

[S39] N. Almarimi, A. Ouni, and M.W. Mkaouer, "Learning to detect community smells in open source software projects," *Knowledge-Based Systems*, Vol. 204, 2020, p. 106201. [Online]. https://www.sciencedirect.com/science/article/pii/S0950705120304226

[S40] M. Hosking, *Transformation troubles and non-technical debt*, 2017. [Online]. https://www.linkedin.com/pulse/transformation-troubles-non-technical-debt-matt-hosking/

e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240102, DOI: 10.37190/e-Inf240102

Continuous Software Engineering Practices in AI/ML Development Past the Narrow Lens of MLOps: Adoption Challenges

Sini Vänskä*, Kai-Kristian Kemell**, Tommi Mikkonen***, Pekka Abrahamsson****

*Deloitte, Finland
**Department of Computer Science, University of Helsinki, Finland
***Faculty of Information Technology, University of Jyväskylä, Finland
****Faculty of Information Technology and Communication Sciences, Tampere University, Finland

sini.vanska@deloitte.fi, kai-kristian.kemell@helsinki.fi,
tommi.j.mikkonen@jyu.fi, pekka.abrahamsson@tuni.fi

Abstract

Background: Continuous software engineering practices are currently considered state of the art in software engineering (SE). Recently, this interest in continuous SE has extended to ML system development as well, primarily through MLOps. However, little is known about continuous SE in ML development outside the specific continuous practices present in MLOps.
Aim: In this paper, we explored continuous SE in ML development more generally, outside the specific scope of MLOps. We sought to understand what challenges organizations face in adopting all the 13 continuous SE practices identified in existing literature.
Method: We conducted a multiple case study of organizations developing ML systems. Data from the cases was collected through thematic interviews. The interview instrument focused on different aspects of continuous SE, as well as the use of relevant tools and methods.
Results: We interviewed 8 ML experts from different organizations. Based on the data, we identified various challenges associated with the adoption of continuous SE practices in ML development. Our results are summarized through 7 key findings.
Conclusion: The largest challenges we identified seem to stem from communication issues. ML experts seem to continue to work in silos, detached from both the rest of the project and the customers.

Keywords: artificial intelligence, machine learning, continuous software engineering, continuous star, multiple case study

1. Introduction

Continuous Software Engineering is the current trend in Software Engineering (SE). In brief, continuous SE comprises various so-called continuous practices, which aim to eliminate discontinuities in SE in order to make it a more continuous process. For example, continuous integration focuses on closer collaboration between development and deployment.

Submitted: 9 Mar. 2023; Revised: 20 Aug. 2023; Accepted: 21 Aug. 2023; Available online: 28 Aug. 2023

Continuous SE is currently state of the art in SE, and companies even consider certain continuous practices vital for remaining competitive going forward [1].

In particular, bridging the gap between development and operations has received much attention following the advent of continuous SE. Indeed, one widely discussed topic related to continuous SE is DevOps. DevOps, a portmanteau of Development and Operations, focuses on the integration between development and deployment. It involves a number of continuous SE practices, which are also arguably some of the most high-profile ones: continuous integration, continuous delivery, and continuous deployment, as well as testing automation (or continuous testing), are typically considered to be necessary for DevOps [2]. To further highlight the current importance of continuous SE practices, we turn to Moreschini et al. [3] who state that "DevOps practices are the de facto standard when developing software".

By now, continuous SE practices have been explored in various contexts in SE research, including the field of Machine Learning (ML). In ML system development, continuous SE is still an emerging phenomenon. In fact, ML system development in general remains a novel topic from the point of view of SE [4, 5]. Much of the current discussion on continuous SE in the context of ML has been focused on the concept of MLOps. With some simplification, MLOps can be considered to be the application of DevOps into the context of ML systems [6].

Continuous SE, as Fitzgerald and Stol [7] conceptualize it, however, is a much larger phenomenon than DevOps (or MLOps), which only comprises a small set of continuous SE practices. Fitzgerald and Stol [7] posit that continuous SE comprises at least 13 different continuous practices, split between the areas of business, development, operations, and innovation. Many of these other continuous SE practices have received little attention compared to the ones included in DevOps. This is especially the case for ML development, where little research on continuous SE exists outside the topic of MLOps. Moreover, MLOps in and of itself is still an emerging phenomenon [6].

Overall, the development of ML systems presents novel challenges in SE, as ML components need to be incorporated into the software system being developed. These components require new know-how and are typically developed separately from the rest of the system by different personnel (e.g., data scientists). Incorporating the development of ML components into the general SE process (via tooling, methods, practices, etc.) is the core issue behind these challenges from the point of view of SE [4]. While ML has been widely studied across disciplines, much of this extant research has focused on technical challenges in ML instead of looking at the development of these systems through the lens of SE [4]. This is especially the case for continuous SE in ML, where little research outside the context of MLOps exists.

In this paper, we begin to address this perceived gap by studying ML development from a more general continuous SE point of view. In doing so, we look past the lens of MLOps (and DevOps) in order to explore the thus far largely unexplored (in ML development) continuous SE practices described by Fitzgerald and Stol [7]. We begin to explore this topic using an exploratory, qualitative multiple case study ($n = 8$) research approach. The specific research question we tackle is formulated as follows: *what are the challenges associated with the use of continuous SE practices in ML development?* Additionally, we are interested in understanding which continuous SE practices are currently used in ML development, which are not, and why.

2. Background

In this section, we discuss the theoretical background of this study. In Section 2.1, we discuss continuous SE in more detail. In Section 2.2, we discuss ML development and how it relates to conventional software development. In Section 2.3, we discuss related work on MLOps and challenges in ML development.

2.1. Continuous SE

Continuous SE aims to eliminate discontinuities in SE [7]. While previous lightweight methodologies have already stressed the importance of focusing on error detection and quick fixes in particular, continuous SE is more holistic. Indeed, in continuous SE, the entire software life cycle, including business strategy and operations, is considered to be a part of development process [7]. Continuous SE builds on agile SE, and agile has been argued to be an important requirement for adopting, e.g., DevOps [8].

Continuous SE, in practice, encompasses various different practices that Fitzgerald and Stol [7] refer to as continuous* (continuous star). These, they argue, can be split into three areas (as seen in Figure 1) that together comprise continuous SE: (1) Business Strategy and Planning; (2) Development, and (3) Operations. In addition (continuous) improvement encompasses all three. Whereas DevOps focuses on collaboration between development and operations, collaboration between business strategy and planning and development is referred to as BizDev, and collaboration between all three is referred to as BizDevOps [7].

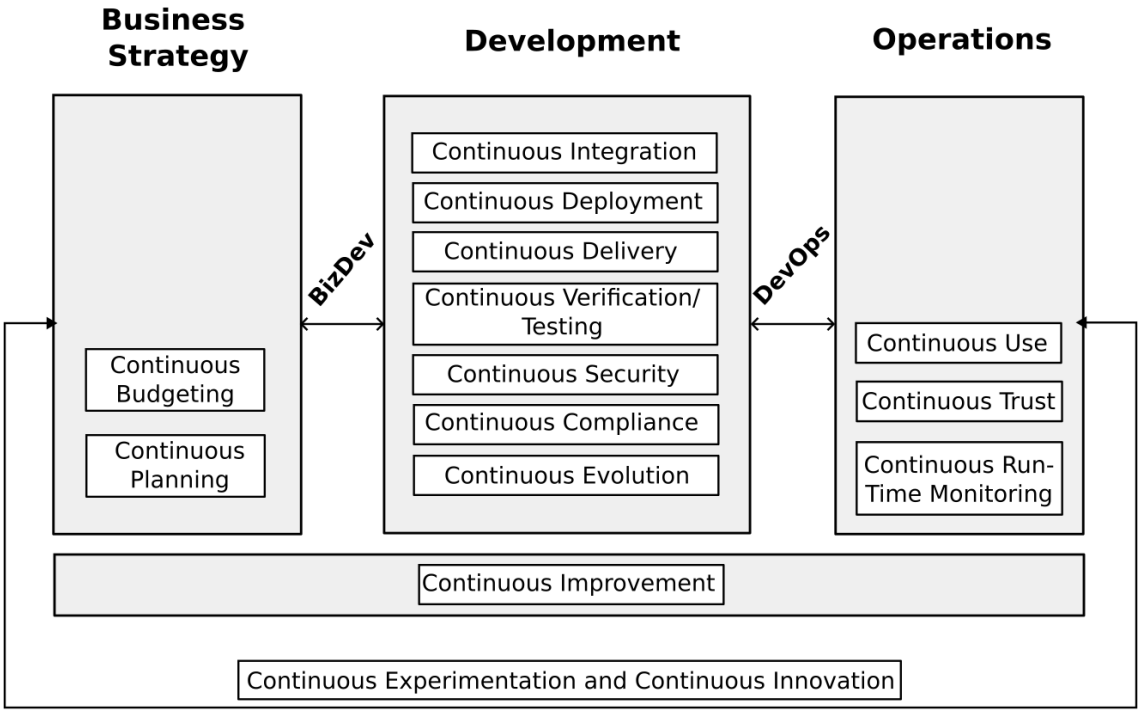


Figure 1. Continuous SE according to Fitzgerald and Stol [7]

Altogether, 13 different continuous* practices are identified by Fitzgerald and Stol [7]. These 13 practices are as follows. The first area (business strategy and planning)

includes continuous planning and continuous budgeting. The second area (development) includes continuous integration, continuous deployment, continuous delivery, continuous verification/testing, continuous security, continuous compliance, and continuous evolution. The third area (operations) includes continuous use, continuous trust, and continuous run-time monitoring. Finally, continuous improvement encompasses all the areas, and continuous experimentation and innovation drive organizations to perform better [7]. These are all illustrated in Figure 1.

Continuous SE further reinforces the need to break down silos inside organizations. DevOps focuses on breaking down the silos between development and operations [9]. However, even more comprehensive collaboration between departments may be needed to achieve continuous SE in some contexts [10]. For example, in the context of ML development, continuous SE faces new challenges as the continuity needs to be extended to also include the ML experts working on the ML components of the system [6], as we discuss in Section 2.3. Next, in Section 2.2, we discuss what makes ML development different from the point of view of SE.

2.2. ML Development from the point of view of software engineering

AI has largely become synonymous with ML today [5]. ML applications are useful for poorly understood problem domains, domains with valuable regularities in their databases waiting to be discovered, and domains in changing environments [11]. In terms of SE practice, ML systems differ from conventional software systems due to the addition of ML components. Developing these components requires new know-how and is handled by various ML experts (e.g., data scientists and ML developers), who are a new addition to the SE process [5]. Aside from various the technical challenges associated with developing ML components, integrating the ML experts into the larger SE workflow poses challenges for organizations developing ML [4].

ML systems are often divided into three classes. The most common is supervised learning where the training data and the "right answer" are accessible. In unsupervised learning, the systems learn by trying to find the common structure in the data on their own. The third, so-called reinforcement learning, refers to systems that evolve by learning in a sequence that leads it to a given goal [12]. The process requires a lot of testing and data sets created (or simply used) for training purposes, and the product can still fail to fulfill its requirements. Moreover, resource estimation in ML development is difficult [13].

Yet, as some existing papers argue, much of ML development is ultimately still software development. Mikkonen et al. [12] cite Google in stating that "even if ML is at the core of an ML system, only 5% or less of the overall code of that total ML production system" is code related to ML. Indeed, ML features are simply "embedded into a larger software system that hosts, provides access to, and monitors ML features" [10], while the rest of the development endeavor is conceptually speaking conventional SE.

Nonetheless, the addition of these ML components into the SE process poses challenges from the point of view of SE. Perhaps most importantly, as already touched upon, the ML development requires collaboration between ML experts and rest of the development team(s) [4, 5]. Two studies reviewing existing literature (one systematic literature review [4] and one systematic mapping study [5]) highlight various challenges in ML development discussed by existing literature. Martínez-Fernández et al., in their systematic mapping study [5], list various challenges associated with SE for ML systems, highlighting the wide variety of challenges associated with ML development from a SE point of view. These include

challenges such as end-to-end ML-based systems including components written in a wide variety of programming languages, reliance on third party components, ML presenting challenges from the point of view of changing requirements (whereas ML components create entanglement), etc.

Giray [4] conducted an SLR specifically focused on SE challenges in ML development, identifying various challenges unique to ML system development across the following six main categories: (1) requirements engineering (RE), (2) design, (3) software development and tools, (4) testing and quality, (5) maintenance and configuration management, and (6) SE process and management. First, in relation to RE, issues such as difficulties managing customer expectations due to lack of customer ML understanding, quantitative measurements being new to many stakeholders, and having to deal with new types of quality attributes (e.g., explainability, fairness) resulted in challenges. Secondly, design-wise, ML systems posed challenges from the point of view of monitoring performance degradation on production (concept drift, etc.), as well as in terms of system architecture, as the interplay of the different parts of an ML system can often result in issues. Thirdly, the vast number of tools and dealing with development environments and infrastructure, both in terms of understanding them and simply successfully utilizing them in practice (compatibility, etc.), were among the key challenges related to software development and tools, in addition to issues related to dealing with ML models and the data required for ML. Fourthly, testing ML components remains a challenge due to the non-deterministic nature of ML systems, including issues related to designing and evaluating test cases, preparing test data, executing tests, and evaluating test results, as well as actually fixing any bugs that are found (which may also be bugs in the ML libraries, frameworks, and platforms being used). Fifthly, dealing with a history of experiments, re-training and re-deployment, and the configuration management (CM) of data and ML models present new challenges for maintenance and CM. Finally, harmonizing the activities for developing ML components with the rest of the software development process, assessing the ML process, and estimating effort are new challenges related to the SE process and management in ML development.

2.3. Related work: MLOps and continuous SE in ML

Thus far, studies on continuous SE in ML have predominantly approached the topic through MLOps. MLOps can be largely considered DevOps for ML [6, 14]. John et al. [6] conceptualize MLOps in their study, highlighting three different pipelines (data, model, and release), each with 3–4 subprocesses, that together comprise the MLOps pipeline.

Lwakatare et al. [15] discuss SE challenges associated with DevOps in ML development contexts, focusing on technical aspects such as issues related to data and ML models. Symeonidis et al. [16] discuss tooling for MLOps and challenges related to it. Granlund et al. [17] discuss challenges related to implementing MLOps practices in a highly regulated application domain (medical). A number of other papers on MLOps are listed in the multi-vocal literature of John et al. [6], and in the systematic literature reviews of Kolltveit and Li [14] and Lima et al. [18]. Overall, much of the existing research on MLOps has focused on defining the concept and on challenges related to establishing an MLOps pipeline (e.g., challenges related to tooling). Fewer papers can be found on human aspects (e.g., project communication, etc.).

As our focus is on challenges, papers related to challenges in ML development could also be seen as related work. Indeed, we find connections between our results and such papers in Section 6. In this regard, a paper by Serban and van der Blom [19] looks at

SE best practices for ML development in general and discusses the adoption rates of the practices, pointing towards challenges in some areas. The secondary studies of Giray [4] and Martínez-Fernández et al. [5] also list various SE challenges related to ML development based on existing literature, as we have discussed in Section 2.2. Other primary studies discussing challenges include the study of de Souza Nascimento et al. [20] that looks at challenges in ML development overall, with the main challenges being: identifying the clients' business metrics, lack of a defined development process, and designing the database structure. Nahar et al. [21] specifically look at communication and collaboration challenges between data scientists and software developers in ML development in their empirical study.

On the other hand, we are unable to identify existing empirical papers discussing continuous* more generally for ML development. As established in Section 2.1, MLOps (and DevOps) only cover some of the 13 continuous* practices discussed by Fitzgerald and Stol [7]. Thus, in looking at continuous* more generally in this study, we believe this paper presents a novel contribution in the area with its point of view.

3. Research framework

This study is an empirical study utilizing qualitative, thematic interviews as the data collection method (see Section 4). Prior to the data collection, to aid in the creation of a suitable interview instrument, we constructed a research framework. This research framework is split into five elements. Fitzgerald and Stol [7] classify continuous* practices into categories as follows: business, development, operations, and innovation. These are elements (2), (3), (4), and (5) in our framework. In addition, we look at current tools and the use of methods (1) related to both continuous SE and, more broadly, agile in ML development, to better understand the development practices being utilized. These five elements are described in more detail below:

(1) Current tools and use of methods. The first part of the research framework is focused on understanding the tools and methods used in ML development. In particular, this part of the research framework is focused on collecting data on development tools and practices, and especially continuous or agile frameworks or methods used in ML development.

(2) Business strategy and planning. Continuous SE considers software development a continuous, holistic process that bridges different organizational units, and also consequently encompasses business strategy and planning [7]. This link between development and business is referred to as BizDev (which in turn is a part of BizDevOps). The purpose of this theme is to understand how ML experts see their organizations' business and strategic elements. This component is also related to understanding how ML experts deal with changing customer/business requirements.

(3) Development. This part of the research model aims to understand the ML development actions, and how they relate to the project as a whole, i.e., whether the ML related tasks are integrated into the rest of the development process, or more of an independent, separate, and siloed process.

(4) Operations. Continuous SE also includes bridging together development and operations (as in DevOps). This part of the research framework is focused on understanding what practices are used after the project goes into production and how the system is monitored past this point. Existing DevOps literature argues that agile transformation is essential for improving the efficiency of the company in optimizing the lifecycle delivery,

breaking the gaps, and creating a continuous feedback loop between the business users and the development teams [8].

(5) Innovation. Fitzgerald and Stol [7] consider continuous improvement and innovation a part of continuous SE. In continuous SE, a new planning phase starts when new opportunities are recognized. This part of the research model is focused on understanding how ML experts feel about new opportunities, innovation, and new technologies. E.g., are they motivated to suggest new ways of developing ML systems based on lessons learned, both in terms of technologies and project management? In this regard, the state of their current project(s) was discussed in relation to what they would do next.

4. Research method

In this section, we discuss the methodology of the empirical study we conducted. In Section 4.1, we discuss our data collection approach. In Section 4.2, we discuss our data analysis approach.

4.1. Data collection

The data for this study were collected through qualitative interviews. The interviews in question were thematic, semi-structured interviews. The themes for the interviews were based on the research framework discussed in the previous section, i.e., there were five themes that the questions were focused on: (1) current tools and use of methods, (2) business strategy and planning, (3) development, (4) operations, and (5) innovation. The interview instrument in its entirety can be found in the Appendix A.

Eight interviews were conducted with respondents from different organizations working on AI-related projects. The respondents (and thus cases) were selected through convenience sampling. The participants had varied job titles, ranging from research assistant to service manager (see Table 1). Similarly, their past job experience varied greatly, ranging from some months to decades of working with ML technologies. We preferred to have respondents from a variety of organizations, as opposed to more in-depth case studies of fewer organizations, due to the novelty of the topic. Similarly, we opted for a semi-structured and thematic interview format due to the novelty of the research topic.

The interviews were conducted digitally due to the COVID-19 pandemic situation that was on-going at the time of the interviews. The interviews were conducted either in Finnish or English, with the (pre-planned) questions being the same in both cases. Due to the semi-structured approach, each interview was nonetheless different, as additional questions were posed based on the responses of each respondent.

Table 1. Respondents

Respondent	Job title
Respondent 1	Research assistant
Respondent 2	Data scientist
Respondent 3	Research assistant
Respondent 4	Service manager
Respondent 5	Professor (SE)
Respondent 6	Senior lecturer (SE and applied AI)
Respondent 7	Regulation specialist
Respondent 8	Software developer

The average duration of the interviews (recorded data) was 35 minutes, i.e., not including introductory statements, instructions, and any post-interview discussion. The interview recordings were transcribed, and the resulting transcripts were analyzed for this paper. We discuss our analysis approach next.

4.2. Data analysis

To analyze the data, we utilized qualitative thematic analysis as the primary data analysis method. More specifically, we utilized deductive coding as our coding approach. In deductive coding, codes are pre-determined based on a framework and then applied to the data in the coding process.

We utilized deductive coding as the coding approach, using our research framework (see Section 3) as the basis for coding. In practice, the codes were the 13 continuous* practices discussed by Fitzgerald and Stol [7]. As we were interested in exploring how different continuous* practices are utilized in the context of ML development, we focused on determining which practices were utilized and how (if at all), resulting in a rather straightforward coding approach.

These codes were then organized in themes according to our research framework. Thus, these 13 codes were arranged into 5 themes for reporting as follows: (1) current tools and use of methods, (2) business strategy and planning, (3) development, (4) operations, and (5) innovation. These codes and their occurrences are detailed in Table 2. The results of this analysis are reported next in Section 5.

In addition to this coding process focusing on solely continuous* practices, the data was analyzed more generally through the five elements of the research framework. For example, we were interested in the use of tools, which were outside the scope of the continuous* practices used as a framework for deductive coding. This was also the case for agile in relation to continuous* in ML.

Table 2. Codes and their occurrences in the data

Code [Continuous…]	Occurrence(s)
Planning	4
Budgeting	2
Integration	1
Delivery	2
Deployment	3
Verification	2
Testing	2
Compliance	0
Security	0
Evolution	5
Use	1
Trust	0
Run-time monitoring	2
Improvement	1
Innovation	2
Experimentation	3

5. Results

In this section, we discuss the results of the empirical study. This section is split into five subsections, with each subsection covering one of the five themes of our research framework.

While reporting our results, we structure the discussion through the use of **PEC**s (Primary Empirical Contributions). These PECs are intended to communicate our key findings in a clear manner. Later, in Section 6, these PECs are also used to structure the discussion of our results.

Additionally, although we use direct citations from the interviews in the text, it should be noted that these PECs are not based solely on the few citations found in the text. While including all relevant citations would not be feasible in the interest of space, we nonetheless use some citations to liven up the text and to provide some transparency in terms of our use of our data.

First, before going into more detail in our analysis, we can already make one interesting observation based on the codes and their occurrences (Table 2): *continuous compliance* and *continuous security* were not present in our data. In addition *continuous trust* also did not appear in our data the way it is understood by Fitzgerald and Stol [7] (who consider it to mean trust developed over time based on the belief that customer expectations are fulfilled without exploiting their vulnerabilities). In our data, only one of the respondents discussed *continuous trust*, and did so on a more general level in relation to having a good relationship with the client.

PEC1: Continuous compliance, continuous trust, and continuous security were not present within the data.

5.1. Overview of used tools and methods

The first interview theme (out of the five discussed in Sections 3 and 4) was focused on the current work role of the respondent and the tools and methods used in the ML projects they were involved with. The tools of interest included programming languages and software tools used in ML development, alongside any other tools the respondents considered relevant enough to mention.

Python was by far the most common language, discussed by seven of the eight respondents. Java was discussed by two. In terms of software tools utilized, the answers of the respondents were more diverse, especially regarding database systems and cloud services. For example, Respondent 2 discussed Dataprix, which was not brought up by any other respondent:

[Development was] 99% or like fully Python based, but then some of the data preparation was done in a Dataprix, you know the kinda Spark service, so that was used. [...] *I would not say that I followed any framework with intent but rather trying to have like the mindset within many of the frameworks.* [R2]

For the most part, the respondents discussed choosing tools based on their own preferences. Some respondents mentioned having a background in ML as a hobby and, thus, opted to use the tools they were familiar with. Most respondents worked either alone or in a small team with other ML experts, and consequently they were the only ones who needed to understand the tools and the ML components and code. On the other hand, some organizations had processes in place which determined what tools should/could be used. Nonetheless, little consensus existed in terms of tooling.

Aside from tooling, we also explored the use of SE methods. The respondents mentioned Scrum, SAFe, DevOps, and MLOps when asked about the use of methods and other development processes. However, the respondents discussed their use critically. The respondents seemed to consider it more important that, as opposed to strictly following a method, practices that fit the current project context are used on a case-by-case basis. Moreover, in this regard, there seemed to be a disconnect between the rest of the organization and the ML experts. For example, Respondents 2 and 4 both mentioned that their organization had adopted an agile approach but that it had little impact on their ML development aside from it being centered around sprints as a result.

The work is planned in sprints. We have 4 planning periods per year. If we speak about doing AI then you can think that sprints always produce something that can be put into production, with the next sprint it will be improved and expanded. [R4]

Some respondents felt that they did not use any methodologies in their development, but rather, a mindset built on many different methods. This seemed common, with many respondents mentioning frameworks but also specifying that they did not use any of them strictly (e.g., ScrumBut), either on the level of the entire organization or just the ML experts.

PEC2: The SE methods used by the organization at large seem to often have little impact on the ML development processes.

5.2. Business strategy and planning

Questions related to this theme were focused on teamwork, requirements, and resources. In both continuous* and agile, collaboration between business and development teams is considered important.

Half (4) of the respondents said that the ML experts in their organization mainly worked independently. In these cases, only project planning activities (e.g., sprint planning) and result reviews were carried out with the rest of the project team, while most of their work was carried out independently. Nonetheless, the majority of the respondents (6) considered collaboration on a project-level important when discussing teamwork and planning. In particular, two respondents working in large corporations with highly regulated projects considered collaboration and communication a critical success factor for producing ML components that fulfilled their requirements. Yet, aside from one respondent who worked on a project with internal stakeholders, the respondents felt that their work was very independent with little collaboration with other employees. To this end, three respondents felt that they knew very little about the work of other people inside the organization as well.

Multiple respondents discussed issues related to collaboration, or from the point of view of continuous*, *continuous planning* (and budgeting, though hardly discussed). To some extent, this lack of collaboration was attributed to the remote work situation stemming from the COVID-19 pandemic situation that was on-going at the time, although not all respondents worked remotely. Respondent 3 highlighted the importance of informal communication for formulating new ideas and discussing problems with co-workers, while otherwise collaboration was minimal. In fact, three respondents felt that they did not even understand the point of the project they were working on and were simply executing tasks:

Personally, the point of the project is unclear to me. [R3]

Customer involvement was also highly varied between projects. In projects with external customers, customer involvement was not regular from the point of view of the ML experts. Most felt that they were working in a silo, especially as far as the customer was considered.

The respondents also discussed challenges related to *(continuous) budgeting*, as well as resource allocation more generally. Multiple respondents mentioned working on multiple projects at a time. Respondent 4 specified that their organization had difficulties finding ML development capabilities, leading to the existing ML experts being stretched thin.

Practically everyone is doing multiple projects. All people are caught in all cases, work needs to be prioritized. [...] *Resourcing is challenging.* [...] *Plus, there is less and less expertise. AI needs to have analysts and people who know algorithms, it is not that simple to have them on every branch.* [R4]

I do not know how they [resources] are decided. It feels like my time is being spent on everything else that is not related to my work. [R3]

As all except one respondent worked on externally commissioned projects, *continuous budgeting* was up to the customer(s). While the respondents had some control over suggesting the addition of new features (or requirements) into the system/project, the customer had the final say in such matters. Some of the projects had additional resources reserved for use in case there were changes in the project scope (e.g., due to added functionalities).

If we get a green light from a company, the project price already includes a lump sum of money either from the company or in the form of some collaboration. And we try to do our best with that or go as far we can go with that sum of money. [R5]

On the other hand, three respondents felt that the customers' indecisiveness caused issues and unpredictability in the development. This, they felt, was frustrating because the customers seemed to not have a sufficient grasp of the realities of ML development, leading to unreasonable demands at times.

For example, the clients might change their mind in every two weeks as it has happened in some projects or the approach to gain some insight to something has changed. [R1]

Overall, the relationship of the respondents and the rest of their organizations, as well as their clients, was vague. The ML experts were not actively involved in project planning and seldom interacted with the customer(s). This resulted in various issues, as discussed above.

PEC3: ML experts seem to often work in a silo. They do not often participate in business strategy related activities.

5.3. Development

The interview questions related to ML development included questions related to how functionalities are added, how the product is tested, and when the product is ready for production. The disconnects between the ML experts and the rest of the project participants were even more apparent in the respondents' responses when discussing practical development matters. When asked about integration, continuous approaches were not discussed by most respondents. Respondent 3 summarized their issues with the lack of collaboration in terms of development as follows:

I feel that it is up to you to decide [when to implement your work] because there is no teamwork, therefore others have nothing to say. I don't know if it's because of my own experience, but it's hard to trust that that product will work. [R3]

While testing was discussed with all respondents, only one respondent [R7] discussed *continuous testing* in the form of automated testing using an MLOps pipeline. Other respondents said that they tested ML functions manually for various reasons. One respondent specified that they felt that automated testing tools for ML were simply not available for their particular system context. Another respondent added that ML functionalities

were difficult to test due to the high level of technical know-how required to develop them, resulting in the tests being carried out by the same individual(s) who developed the functions.

It was not automated for sure. It was all manual. There was unit testing, then there was regression testing, also integration testing and all the things that you see. [R5]

You work with simple test cases at first and apply it into a bigger chunk of data. Then there's some peer review done by the other data scientist in the product team but it kinda depends how well that can work because if you are working with something that is pretty exotic then not even other data scientist might know that much about it. [R2]

PEC4: Automated testing in ML development remains a challenge for organizations developing ML systems.

Continuous verification was mostly discussed in relation to regulations. This was mostly in relation to data, with regulations such as the GDPR (the EU's General Data Protection Regulation) often affecting ML development due to their use of large amounts of data.

Continuity in general seemed to be less of a concern in situations where the organization simply developed a narrow ML functionality or a model that the customer then implemented and monitored on their own. In such situations where no full, functional system was developed, *continuous quality*, for example, was of little concern to the respondents. This also seemed to apply to *continuous verification* and *continuous compliance*.

5.4. Operations

The questions about operations focused on what happened to the product and project after initial deployment. This included usage and monitoring of the product, as well as customer relations. The questions dealt with user interaction, user expectations, monitoring, as well as the conclusion of the project.

Continuous use focuses on understanding whether user expectations are fulfilled. However, the respondents felt that the users were not well understood. The ML experts seldom had direct contact with the user(s) and only dealt with the representatives of the customer organization. In some cases the respondents had no contact with the customer at all and only communicated with their own organization's project staff. When the ML experts got feedback from the users, it was gathered by the customer organization and forwarded to them. No direct channels existed. One of the respondents specifically remarked that such feedback only reached them if something was "great or terribly wrong."

[Interaction with the users] is rare, a privilege. [R6]

With internal customers, interaction was more frequent (as discussed by Respondent 4). With external customers, much depended on who was the end-user and how the project was organized. On the other hand, some ML experts did not want to interact with the user or customer in the first place:

I guess some people like it and it can give some valuable feedback for the developers, but I did not appreciate it in the past. [R2]

PEC5: The lack of user and customer interaction makes it difficult for ML experts to ensure that the product can be continuously used.

Monitoring practices varied between projects. Monitoring was largely left to the customer. In some cases, the ML experts were in charge of monitoring and problem detection right after initial release, until the customer later took over. Active communication with the customer would typically end after the product was delivered. This also meant that operations was not of much concern in such projects from the point of view of the ML experts.

Well, there is a short price/crisis period during which I can still provide help if still needed but it is the customer's job to deal with the rest. [R1]

None of the respondents mentioned trust (*continuous trust*) as something that they tried to actively establish with the customer. The ML experts seemed to nonetheless recognize the importance of having a good relationship with the customer, e.g., in terms of securing future projects.

There are less opportunities to do something completely innovative once you've released the project or the product. But if new opportunities arise in the sense, if it is a long-term relationship with the client, and if you're continuously working on something bigger, then of course you have the possibility of improving or actually innovating or completely replacing something that you've already delivered a couple of years ago. [R7]

The lack of interaction between the ML experts and the users of the product is challenging for operations. Project contracts typically determined what kind of operational activities the ML experts had in that project. The ML experts themselves, however, seemed to not mind this situation. In fact, many of the respondents seemed satisfied about not having to interact with external stakeholders after deployment.

PEC6: The emphasis placed on operations varies greatly depending on project context. If the customer takes over entirely after release, there is little need for operations from the development organization.

5.5. Improvement and innovation

Due to the disconnect between development and operations resulting from externally commissioned projects and their contractual agreements, only R4, whose organization developed ML systems for internal use, engaged in *continuous improvement and innovation*. As the other organizations (four out of eight) largely moved on after deployment, in some cases following an initial phase where they continued to make sure the system worked as intended briefly after deployment, opportunities for *continuous improvement* were limited. Only if the client proposed further improvements, and provided the funding for them, would work on the system continue past bug fixes or minor improvements made shortly after initial deployment, based on contractual obligations. R4, on the other hand, who worked for internal customers, discussed how feedback gathered after deployment was used to continuously improve the system.

Because the customers were in charge of the projects, improvements and innovations were not automatically thought of, or even considered welcome in the project. As the contracts typically determined the project scope, any potential changes would have to be discussed with the customer. Three respondents described cases where they had proposed small improvements. However, larger innovations were often seen as risky by the customer and the ML experts, and were seldom suggested to the customer. Thus, *continuous innovation* on a project level was largely not considered relevant by the respondents.

On the other hand, half of the respondents felt that an innovative mindset was essential in their line of work on a personal level. As ML is a quickly evolving field, they felt that they needed to be open to learning new things and coming up with new ideas. Even if the current project was easier to carry out as planned earlier, new innovations could help with future ones.

PEC7: New ideas or innovations are not automatically added to on-going ML projects when an external customer is involved, as the customer sets the scope of the project.

6. Discussion

In this section, we discuss the implications of our results. In Table 3 we summarize the Primary Empirical Contributions (PECs) we highlighted during our analysis. We use these PECs to structure the discussion in this section. After covering all the individual PECs, towards the end of this section we return to the research question we outlined at the start of this paper and answer it based on our findings.

Table 3. Primary empirical contributions of the study

PEC	Description
1	Continuous compliance, continuous trust, and continuous security were not present within the data.
2	The SE methods used by the organization at large seem to often have little impact on the ML development processes.
3	ML experts seem to often work in a silo. They do not often participate in business strategy related activities.
4	Automated testing in ML development remains a challenge for organizations developing ML systems.
5	The lack of user and customer interaction makes it difficult for ML experts to ensure that the product can be continuously used.
6	The emphasis placed on operations varies greatly depending on project context. If the customer takes over entirely after release, there is little need for operations from the development organization.
7	New ideas or innovations are not automatically added to on-going ML projects when an external customer is involved, as the customer sets the scope of the project.

PEC1, on a more general level, highlights that some continuous* practices receive less attention than others, as seen in more detail in Table 2. This, to some extent, supports our original motivation behind the study: some continuous* practices are seldom studied compared to the most commonly discussed ones. However, as our data set is not large, given the exploratory nature of the study, we would not place much emphasis on this particular observation. It is also worth keeping in mind that the domain of the project may play a large role in how relevant various regulations are (e.g., medical domain) from the point of view of continuous compliance, and that issues such as cybersecurity may be delegated to specific experts within an organization. PEC1 may nonetheless be of interest from the point of view of future studies, however, as it may give an idea of which continuous* practices are common out on the field and which are not.

PEC2. Based on our data, ML development is seldom carried out using SE methods by the book. This is consistent with existing research where the lack of a defined development process in ML development is identified as an issue [20]. Moreover, some of the respondents discussed using some practices, such as sprints, as a part of the project in general, but that 1) the associated method (e.g., SCRUM) was not followed by the book in the project in general, and 2) they were at best erratically applied to the ML portion of the project. PEC2 in this fashion highlights one way in which ML experts continue to work in a silo (as they often seem to do [22] at present). As far as agile approaches are considered, our results support the observations of Serban and van der Blom [19] who posit, based on a survey, that traditional SE practices have a lower adoption rate than ML specific practices in ML development. Our results in this regard provide further insights on how this manifests in practice. Finally, the respondents of our study discussed a wide variety of tools (database systems, cloud services, etc.) used in ML development, which corresponds with existing

research where, e.g., Kim et al. [23] highlight that the vast number of available tools can be a challenge in and of itself.

PEC3 corresponds with extant research in that bridging the gap between the ML experts and the rest of the developers is a challenge in ML development [4, 21]. It seems common for ML experts to work in a silo [21, 22], although there are also examples of successfully integrating ML experts with the rest of the development team(s) found in existing papers [23]. This is also an issue MLOps aims to tackle as an approach to ML development, much like how DevOps focused on bridging the gap between development and operations. Our findings may help further illustrate what this means in practice and what kind of issues this results in.

Extant research argues that one of the key factors of success in implementing DevOps is communication [24]. This, thus, presents various problems for adoption of continuous* as well, as DevOps belongs under the umbrella of continuous*. Communication issues in ML development are also acknowledged in extent literature, where, indeed, communication issues between the ML experts and the rest of the development team and organization are considered a recurring challenge [4, 5]. Educating software engineers on ML and ML experts on SE could help in this regard by making it easier for the project participants to develop a mutual understanding of the project [21].

However, communication issues are not an issue unique to ML development or continuous SE in the context of ML. Indeed, organizations adopting DevOps often face issues related to communication in and between teams, due to, e.g., differences in the professional and personal backgrounds of the employees [25]. Issues with communication and siloing are also seen in relation to software security, where collaboration between security experts and software developers has been a challenge, and where more recently DevSecOps has looked to improve the situation in a similar manner to DevOps and MLOps [26]. Just as how ML experts may have problems communicating with software developers and vice versa due to their different areas of expertise [21], security experts and software developers face communication issues as well (e.g., developers may feel attacked when security experts point out security flaws in their code) [26]. In fact, interdisciplinary collaboration between team members with differing academic and professional backgrounds is seen as a challenge in teamwork overall [27].

PEC4 highlights another challenge in adopting continuous practices: lack of experience with testing among ML experts. Only one respondent discussed utilizing an MLOps pipeline for automated testing. The other respondents felt that tooling was still lacking in the area, or were simply not concerned with automated testing. One potential issue here could be the specific know-how required in ML development: the people otherwise in charge of testing in the project may not have the required skillset to carry out the testing on the ML components. This is consistent with existing literature where various challenges associated with testing ML systems are discussed [4]. Serban and van der Blom also report that testing ML artefacts overall (even outside doing so in a continuous fashion) remains a challenge in software organizations based on the low adoption rate of ML testing best practices [19].

PEC5 further points towards communication issues on a project-level (together with PEC3). Many of our respondents felt that they had very little interaction with the (end-)users, or even the customer organization. This further points towards ML experts working in a silo. ML experts seem to often be in a silo not just in relation to other project members within the same organization, but also in relation to the customers and users. Existing research considers identifying the relevant business metrics of the customer

a challenge in ML development, as the customer company itself may not understand what data or metrics could be important [4, 20]. Not having ML experts interact with the customer would seem to be a bad practice, unless there is someone acting as a bridge between the ML experts and the customer who is capable of bridging this communication gap. Multiple existing studies have highlighted challenges associated with requirements engineering in ML development, and many of these challenges are related to communication (e.g., lack of knowledge of ML on the part of the customer(s), dealing with quantitative measurements for requirements, etc.) [4].

PEC6 provides some additional insights into customer involvement in ML project contexts. While customer and user involvement more generally is a widely studied topic in SE, and is at the core of agile SE as well, ML development adds complexity to SE projects in this regard, too. Maintenance of ML systems results in novel challenges in SE [4] as, for example, an otherwise technically functioning system may still degrade in (ML) performance over time due to concept drift [28]. How this is handled in projects where the system is ultimately handed over to a customer is a practical challenge that the customer organizations need to be aware of.

Finally, PEC7 highlights a conflict between practical project matters and organizational/personal interest. The respondents felt that continuous innovation was important, especially on a personal level, in a field as topical and rapidly evolving as ML development. Yet, when ML systems or ML components were developed for an external customer, innovating during projects was not considered beneficial. New innovations would only serve to increase the scope of on-going projects and doing so was seen as counter-intuitive, or simply difficult, because it would necessitate having the customer okay the changes first.

6.1. Answers to research questions

Finally, to directly answer the research question we posed at the start of this paper (i.e., *what are the challenges associated with the continuous development of artificial intelligence?*), we summarize our results as follows. Our findings suggest that the adoption of continuous software engineering in the development of ML has many challenges caused by the addition of ML components into the SE process. ML development is carried out in a more rigid fashion than agile SE in the context of conventional software, and ML experts mainly work independently, i.e., in a silo. This results in difficulties adopting continuous* practices that require collaboration across teams. Communication issues caused by a lack of shared knowledge, lack of guiding frameworks, and issues related to the roles and responsibilities of ML experts meant that the project life cycle did not resemble a continuous cycle but a step-by-step heavyweight development model. Furthermore, the ML experts rarely interacted with the customer or the product users, as they felt that their work role did not include such actions.

Perhaps partially as a result of the types of projects the respondents were involved in, the respondents also discussed challenges adopting continuous* practices related to operations. Many of the respondents worked in projects commissioned by external customers where the customer was largely responsible for the ML system once it had been deployed (or once the ML component had been finished and delivered). Thus, the ML experts had little control over the operational life of the system or components past an initial grace period where it was jointly monitored after release to ensure it worked as intended. Such practical, project-specific challenges require deliberation from the project participants, as

the maintenance (or continuous development) of such systems is important in ML where, for example, concept drift can degrade the performance of a system over time.

While ML is currently coveted by organizations everywhere, few organizations possess ML development competencies themselves. This situation makes externally commissioned ML projects common. Such projects can pose challenges when it comes to utilizing continuous* practices if the monitoring and operations are left to the customer organization. This can lead to continuous* practices receiving less emphasis in organizations working on such ML projects.

6.2. Limitations

We utilized a qualitative thematic interview approach in this study. This study design poses limitations to the results of the study. First, the respondents held different roles in their organizations and these roles influenced their answers. For example, one of the respondents worked in a management role and as such could provide more insights into the business aspects of the project, but was less knowledgeable about the technical aspects, and vice versa in the case of some other respondents. Moreover, as we interviewed only one respondent per organization, we arguably gained a limited understanding of each organization through the lens of a single respondent. We selected this approach due to the novelty of the topic, as we wanted to explore a larger number of organizations to understand how (continuous) ML development is handled in different organizations.

Secondly, in spite of this, the number of the respondents (and organizations, where $n = 8$) is a potential limitation when it comes to generalizing the results of the study. To this end, it is a limitation as well that three of these organizations were research projects with industry collaboration, as opposed to purely industrial contexts. However, given the novelty of the topic, we argue that this is a sufficient number for an exploratory study into a novel topic, e.g., Eisenhardt [29] recommends case study research particularly for novel research topics. Thirdly, we highlight our utilization of convenience sampling as a further limitation for this study, both on the level of organizations and respondents.

Finally, due to the novelty of the topic, we also utilized a more general research approach. We did not focus, for example, only on certain technologies or project contexts. While this approach let us gather more diverse data, this also presents some further limitations for the study. This study ultimately provides a look at the current state of practice more generally, i.e., we studied how different types of organizations develop ML systems. This makes our findings less specific as well. Another approach to the topic could have been to study organizations that specifically (claim to) utilize MLOps, for example, and to discuss what they felt had been the largest challenges in adopting it in the past, or challenges that they still continued to face. In such a fashion, the scope of the study could have been very different. As it is, we have studied what aspects of continuous SE are used in different projects, and which ones are omitted, across a more diverse set of organizations. We chose this approach due to the fact that we specifically wanted to explore less commonly discussed continuous* practices as opposed to the ones present in DevOps and MLOps.

7. Conclusions and future research suggestions

In this paper, we have explored the utilization of continuous* practices (as conceptualized by Fitzgerald and Stol [7]) in the context of ML development. Though continuous SE is

currently state of the art in SE, it has been less studied in the context of ML development. In particular, continuous* practices outside those related to DevOps, and in this case MLOps, have received little attention in ML development.

Through qualitative, thematic interviews with 8 respondents from different organizations involved in ML development, we sought to understand current challenges organizations face in continuous SE in the ML development context. Our findings are summarized in Table 3 at the start of the preceding section. In brief, the largest issues across the board were related to a lack of collaboration and communication between the ML experts and the rest of the project team(s) and stakeholders. With ML experts largely working in a silo, utilizing continuous* practices in the development of the ML components is challenging.

This paper presents a starting point for studying continuous SE in the context of ML development, outside the specific context of MLOps. Much like how DevOps, which it is based on, MLOps only comprises some continuous SE practices. Many of the continuous SE practices discussed by Fitzgerald and Stol [7] are out of the scope of these processes. We hope to encourage further studies into these practices in the context of ML development, as well as in SE overall. Future studies should adopt more specific points of views to the topic in delving deeper into the phenomenon (e.g., by focusing on specific, but nonetheless less commonly studied continuous* practices). The main contribution of this paper is to provide an initial look at the current state of practice through challenges associated with the adoption of these practices.

Acknowledgments

This work was partly funded by local authorities ("Business Finland") under grant agreement ITEA-2020-20219-IML4E of the ITEA4 programme.

References

[1] C. Parnin, E. Helms, C. Atlee, H. Boughton, M. Ghattas et al., "The top 10 adages in continuous deployment," *IEEE Software*, Vol. 34, No. 3, 2017, pp. 86–95.

[2] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of DevOps concepts and challenges," *ACM Computing Surveys (CSUR)*, Vol. 52, No. 6, 2019, pp. 1–35.

[3] S. Moreschini, F. Lomio, D. Hästbacka, and D. Taibi, "MLOps for evolvable AI intensive software systems," in *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 1293–1294.

[4] G. Giray, "A software engineering perspective on engineering machine learning systems: State of the art and challenges," *Journal of Systems and Software*, Vol. 180, 2021, p. 111031. [Online]. https://www.sciencedirect.com/science/article/pii/S016412122100128X

[5] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert et al., "Software engineering for AI-Based systems: A survey," *ACM Transactions on Software Engineering and Methodology*, Vol. 31, No. 2, 2022.

[6] M.M. John, H.H. Olsson, and J. Bosch, "Towards MLOps: A framework and maturity model," in *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2021, pp. 1–8.

[7] B. Fitzgerald and K.J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, Vol. 123, 2017, pp. 176–189. [Online]. https://www.sciencedirect.com/science/article/pii/S0164121215001430

[8] I. Karamitsos, S. Albarhami, and C. Apostolopoulos, "Applying DevOps practices of continuous automation for machine learning," *Information*, Vol. 11, No. 7, 2020. [Online]. https://www.mdpi.com/2078-2489/11/7/363

[9] R.V. O'Connor, P. Elger, and P.M. Clarke, "Continuous software engineering – a microservices architecture perspective," *Journal of Software: Evolution and Process*, Vol. 29, No. 11, 2017, p. e1866. [Online]. https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1866

[10] S. Mäkinen, H. Skogström, E. Laaksonen, and T. Mikkonen, "Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?" in *2021 IEEE/ACM 1st Workshop on AI Engineering – Software Engineering for AI (WAIN)*, 2021, pp. 109–112.

[11] D. Zhang and J.J. Tsai, "Machine learning and software engineering," *Software Quality Journal*, Vol. 11, 2003, pp. 87–119.

[12] T. Mikkonen, J.K. Nurminen, M. Raatikainen, I. Fronza, N. Mäkitalo et al., "Is machine learning software just software: A maintainability view," in *Software Quality: Future Perspectives on Software Engineering Quality*, D. Winkler, S. Biffl, D. Mendez, M. Wimmer, and J. Bergsmann, Eds. Cham: Springer International Publishing, 2021, pp. 94–105.

[13] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Transactions on Software Engineering*, Vol. 21, No. 2, 1995, pp. 126–137.

[14] A.B. Kolltveit and J. Li, "Operationalizing machine learning models: A systematic literature review," in *Proceedings of the 1st Workshop on Software Engineering for Responsible AI*, SE4RAI '22. New York, NY, USA: Association for Computing Machinery, 2023, p. 1–8.

[15] L.E. Lwakatare, I. Crnkovic, and J. Bosch, "DevOps for AI – Challenges in development of AI-enabled applications," in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2020, pp. 1–6.

[16] G. Symeonidis, E. Nerantzis, A. Kazakis, and G.A. Papakostas, "MLOps – Definitions, tools and challenges," in *IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, 2022, pp. 0453–0460.

[17] T. Granlund, V. Stirbu, and T. Mikkonen, "Towards regulatory-compliant MLOps: Oravizio's journey from a machine learning experiment to a deployed certified medical product," *SN Computer Science*, Vol. 2, 2021.

[18] A. Lima, L. Monteiro, and A.P. Furtado, "MLOps: Practices, maturity models, roles, tools, and challenges – A systematic literature review," in *Proceedings of the 24th International Conference on Enterprise Information Systems*, 2022, pp. 308–320.

[19] A. Serban, K. van der Blom, H. Hoos, and J. Visser, "Adoption and effects of software engineering best practices in machine learning," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–12.

[20] E. de Souza Nascimento, I. Ahmed, E. Oliveira, M.P. Palheta, I. Steinmacher et al., "Understanding development process of machine learning systems: Challenges and solutions," in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019, pp. 1–6.

[21] N. Nahar, S. Zhou, G. Lewis, and C. Kästner, "Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 413–425.

[22] D. Piorkowski, S. Park, A.Y. Wang, D. Wang, M. Muller et al., "How AI developers overcome communication challenges in a multidisciplinary team: A case study," *Proceedings of the ACM on Human-Computer Interaction*, Vol. 5, No. CSCW1, 2021.

[23] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "Data scientists in software teams: State of the art and challenges," *IEEE Transactions on Software Engineering*, Vol. 44, No. 11, 2017, pp. 1024–1038.

[24] L. Riungu-Kalliosaari, S. Mäkinen, L.E. Lwakatare, J. Tiihonen, and T. Männistö, "DevOps adoption benefits and challenges in practice: A case study," in *Product-Focused Software Process Improvement: 17th International Conference, PROFES*. Trondheim, Norway: Springer, 2016, pp. 590–597.

[25] M.S. Khan, A.W. Khan, F. Khan, M.A. Khan, and T.K. Whangbo, "Critical challenges to adopt devops culture in software organizations: A systematic review," *IEEE Access*, Vol. 10, 2022, pp. 14 339–14 349.

[26] N. Tomas, J. Li, and H. Huang, "An empirical study on culture, automation, measurement, and sharing of devsecops," in *International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 2019, pp. 1–8.
[27] S. Brandstädter and K. Sonntag, "Interdisciplinary collaboration: How to foster the dialogue across disciplinary borders?" in *Advances in Ergonomic Design of Systems, Products and Processes: Proceedings of the Annual Meeting of GfA 2015*. Springer, 2016, pp. 395–409.
[28] L. Baier, F. Jöhren, and S. Seebacher, "Challenges in the deployment and operation of machine learning in practice," in *ECIS*, Vol. 1, 2019. [Online]. https://aisel.aisnet.org/ecis2019_rp/163/
[29] K.M. Eisenhardt, "Building theories from case study research," *The Academy of Management Review*, Vol. 14, No. 4, 1989, pp. 532–550.

Appendix A. Interview instrument

Theme 1: Current job and challenges
1. What is your current work role and what does it include?
2. How is the work divided in your project group? How much do you collaborate with others?
3. What kind of tools did you use when developing AI?
4. Can you name any framework, model, or mindset that you used to guide the development process?

Theme 2: Business strategy
1. Can you work independently in the project, or does your work require collaboration with other project participants?
2. How are the requirements of the project decided? You can use a previous or current project as an example.
3. How are the resources planned at the beginning of the project?

Theme 3: Development
1. When is a new functionality or part of the code applied to the larger project context at hand?
2. Can you describe the testing process in some of your projects?
3. How do you decide that the system is ready to be released?
4. Do you know how your work affects the overall quality of the project? For example, the quality of a software project.

Theme 4: Operations
1. Do you interact with the product users after the release?
2. After the product release, do you know if the user expectations are fulfilled?
3. Is the product monitored after the release? If it is, how?

Theme 5: Improvement and innovation
1. Is the product quality improved after the release by you?
2. Are new innovations added to the product if new opportunities arise?
3. When does your involvement with the project end?

e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240103, DOI: 10.37190/e-Inf240103

Software Defect Prediction Using Non-Dominated Sorting Genetic Algorithm and *k*-Nearest Neighbour Classifier

Mohammad Azzeh*, Ali Bou Nassif**, Manar Abu Talib***, Hajra Iqbal****

*Department of Data Science, Princess Sumaya University for Technology, Jordan
**Department of Computer Engineering, University of Sharjah, UAE
***Department of Computer Science, University of Sharjah, UAE
****Department of Computer Engineering, University of Sharjah, UAE

m.azzeh@psut.edu.jo, anassif@sharjah.ac.ae, mtalib@sharjah.ac.ae, u16107036@sharjah.ac.ae

Abstract

Background: Software Defect Prediction (SDP) is a vital step in software development. SDP aims to identify the most likely defect-prone modules before starting the testing phase, and it helps assign resources and reduces the cost of testing.
Aim: Although many machine learning algorithms have been used to classify software modules based on static code metrics, the *k*-Nearest Neighbors (*k*NN) method does not greatly improve defect prediction because it requires careful set-up of multiple configuration parameters before it can be used. To address this issue, we used the Non-dominated Sorting Genetic Algorithm (NSGA-II) to optimize the parameters in the *k*NN classifier with favor to improve SDP accuracy. We used NSGA-II because the existing accuracy metrics often behave differently, making an opposite judgment in evaluating SDP models. This means that changing one parameter might improve one accuracy measure while it decreases the others.
Method: The proposed NSGAII-*k*NN model was evaluated against the classical *k*NN model and state-of-the-art machine learning algorithms such as Support Vector Machine (SVM), Naïve Bayes (NB), and Random Forest (RF) classifiers.
Results: Results indicate that the GA-optimized *k*NN model yields a higher Matthews Coefficient Correlation (*MCC*) and higher balanced accuracy based on ten datasets.
Conclusion: The paper concludes that integrating GA with *k*NN improved defect prediction when applied to large or small or large datasets.

Keywords: software defect prediction, genetic algorithm, multi-objective optimization, *k*-nearest neighbor

1. Introduction

Software systems are integral to daily life [1–4]. As such, the quality of our software systems must be monitored for defects in order to produce perfectly running, defect-free systems and limit software failures [3, 5–7]. A software defect is simply defined as an error, fault, bug, or problem in a computer program [8]. Software defects are caused by errors made during the implementation phase and can lead to software failures when the software

Submitted: 30 Apr. 2023; Revised: 28 Sep. 2023; Accepted: 18 Oct. 2023; Available online: 25 Oct. 2023

program is executed [9, 10]. Studies have shown that such defects lead to decreased customer satisfaction and increased maintenance costs [11]. Therefore, it is essential to locate software defects and identify which modules require repair or retesting before running a program application. Software Detection Prediction (SDP) is the process of predicting defect-prone modules before starting the testing phase [12]. As software complexity increases, the use of SDP in the software development process has become even more crucial. The necessity of having proper SDP models results in improved efficiency, reduced development time, and reducing time spent on testing and error management [4, 5, 13]. Many SDP models were built in the last decades using several different datasets. The most commonly used publicly accessible dataset is the NASA repository, produced in 2005 [14]. Researchers have used the NASA repository with many machine learning models with promising results [15–19]. The most widely used machine learning algorithms include Naïve Bayes (NB), Neural Networks (NN), k-Nearest Neighbor (kNN), Support Vector Machine (SVM), decision trees, etc. [20, 21].

Our focus in this paper is on kNN algorithm because it has less attention form researchers, and it did not offer significant improvements on the prediction accuracy. One reason for that is the large space of configuration possibilities that govern the execution of kNN which includes 1) choosing distance measure, 2) optimal features sets, 3) optimal feature weights, and most notably, 4) the number of nearest neighbors. This resulted in a massive number of configuration possibilities that extend 100,000 possible solutions. Therefore, searching for the best subset of kNN configuration parameters is relatively impossible unless a more robust searching algorithm is used.

This paper applied Non-dominated Sorting Genetic Algorithm (NSGAII) optimization technique to search for the best kNN solution that fits training data [22]. This algorithm finds the best solution amongst a vast space of solution sets or a primary population where an individual is referred to as a chromosome [23], inspired by Darwin's theory of "survival of the fittest." The best individuals have the best probability of being reproduced in the next generation [24]. GA has been widely used to solve complex problems in computing and engineering fields [25]. The GA process consists of four main phases: 1) initialization, 2) selection, 3) crossover, and 4) mutation. In each cycle of GA construction, the chromosomes with high fitness are most likely reproduced to generate new generations. With the help of crossover and mutation, the GA can make a minor modification to the most fitted individuals to generate hopefully better solutions [26, 27].

However, the existing accuracy metrics often present opposite judgments in evaluating SDP models [19, 28]. This means that changing one parameter could improve one accuracy metric while reducing the other accuracy metrics. Therefore, we used a Pareto front multiobjective NSGAII to come up with an optimal solution that improves the overall accuracy of SDP-based kNN without reducing the other accuracy metrics. The effect of these conflicting parameters together has not been examined before. Thus, we propose to see the construction of kNN as a multiobjective optimization problem. This research is driven by the following research questions:

- **RQ1.** Do the use of NSGAII and the proposed solution vector improve accuracy of kNN model for software defect prediction problem?
- **RQ2.** Is there sufficient evidence that the NSGAII algorithm can find the best k value for each module?

To answer these questions, we integrated the kNN algorithm with NSGAII. We proposed a new solution vector that combines two main kNN variables that must be optimized. The first variable is an integer that represents best k value and the second one is a binary

vector of length m, where m is the number of features. Furthermore, since the evaluation measure cannot give indication to the superiority of the generated configuration, we used mutliobjective optimization based on two evaluation measures that are less vulnerable to imbalanced data, namely, Matthew's Correlation Coefficient (MCC) and Balanced Accuracy (BA). In general, the proposed model produced good performance when compared to baseline kNN and other machine learning algorithms.

The rest of the paper is structured as follows. The first section introduced the research and the subject matter of the paper, an overview of the literature available on software defect prediction, the algorithms we tested, and the results of significance obtained from these research papers. Section 2 presents related work. Section 3 presents a problem presentation. Section 4 presents multiobjective optimization algorithms. The research methodology is presented in Section 5. The obtained results are covered in Section 6. Section 7 introduced threats to validity. Section 8 provides a summarized conclusion to the efforts of this study and the future works, along with limitations.

2. Related work

Many machine-learning models have been used to predict the defects in software systems. Shuai et al. [29, 30] Investigated the use of GA with an SVM classifier and particle swarm algorithm for software fault prediction. Another study investigated a novel dynamic SVM method based on improved Cost-Sensitive SVM (CSSVM) optimized by GA [29]. Results demonstrated a high Area Under the Curve (AUC) value for the GA-CSSVM model compared with only SVM or CSSVM. Recorded results were 0.721 with the KC1 dataset, 0.832 with PC1, and 0.897 with the MC1 dataset. The proposed method resulted in an F1 score of 94.88% with the CM1 dataset, 91.89% with KC1, 94.90 with KC3, 99.7% with MC1, and 95.78% with the MW1 dataset. Elish et al. [31] compared the effectiveness of SVM prediction to that of eight machine learning models in predicting defect-prone software modules. Using the PC1, KC1, and KC3 datasets, the findings showed that SVM outperformed other machine learning models. Hammad et al. [32] used the kNN machine learning algorithm to predict faulty software projects. They used public datasets and four different similarity measures, achieving an accuracy of 87.2% using the Euclidean distance measure. The researchers [7, 11, 33] studied the detailed performance analysis of various machine learning classification techniques using publicly available NASA datasets. They found that evaluation metrics like ROC and accuracy are ineffective performance measures because they do not react to class imbalance. In contrast, precision, recall, f-measure, and MCC react to class imbalance. Moreover, Khoshgoftaar et al. [34] worked on a pre-existing model to enhance the performance of Random Forest by applying feature selection to detect software defects. They achieved high accuracies with PC2, PC2, PC3, and PC4 datasets.

Mabayoje et al. [35] focused on understanding the effect of kNN tuning on the abilities of the SDP based on 6 datasets. The result of significance was that the k value should be greater than 1 and the distance weighting improved the predictive performance of the SDP by 8.82%. Iqbal et al. [36] used Use of a feature selection-based ensemble classification framework, divided over four stages, with the framework implemented with and without feature selection. They Found that the framework proposed in the paper could perform better in comparison to other popular classification techniques including kNN, NB, MLP and OneR among others. One issue that the study faced was with regards to imbalance; a solution was not presented in the paper. Jindal et al. [37] proposed a Solution for defect severity

assessment using text mining and machine learning. The results showed that the kNN technique could predict defects across all severity levels, with the performance improving every time the number of corresponding words was increased. The study concluded that the kNN method is best used for defects of medium severity. Ulumi et al. [38] focused on subject of SDP for cross-project domain. The study made use of kNN to fill in missing values from a dataset, followed by classification using the NB or RF methods. Goyal et al. [39] used the kNN regression as opposed to Ordinary Least Square (OLS) parametric regression for defect prediction. They found the kNN regression remains unaffected with increasing number of predictors and provide better performance when using linear regression.

Based on existing research, it appears that using the kNN method in conjunction with other techniques, such as machine learning, can help develop models capable of conducting SDP with higher levels of accuracy and reliability. In addition to SDP for single projects, kNN presents an exciting front for cross-project defect prediction. Without using kNN, this can be a challenging task since the datasets across domains contain many features. An issue that is also faced is data imbalance, which is a significant issue in SDP; one possible solution is the use of resampling techniques, which enhance the accuracy of machine learning classifiers, including kNN.

Above all, there is little research on the kNN efficacy in predicting defect-prone software systems, and it does not offer significant improvements in prediction accuracy. This is due to the multiple configuration possibilities that govern kNN, including 1) choosing distance measures, 2) optimal feature sets, 3) optimal feature weights, and most importantly, 4) the number of nearest neighbors [35, 40]. Because these configuration possibilities result in 100 000 possible solutions, a more robust searching algorithm is needed to identify the best subset of kNN configuration parameters. One possible solution to this problem is to apply the Multi-Objectives Genetic Algorithm (MOGA) optimization technique to search for the best kNN solution that fits training data. This study hypothesized that optimizing kNN with GA would create a more robust algorithm for SDP.

3. Objective functions

Objective function is important termination factor that tells MOGA to stop iteration before reaching maximum number of iterations. Usually, the classical GA uses one objective function that tell us we found the best solution for a problem. In case of SDP studies, the objective function is frequently one of the accuracy measures. However, there are multiple accuracy measures that are used evaluate SDP models where some of them are less informative for the model performance. Since these accuracy measures (a.k.a. objective functions) behave differently we prefer to use multiple objective function functions. The second important question is which reliable accuracy metrics should be used as objective functions. After careful investigation we found that almost all SDP datasets suffer from imbalanced data distribution where number of non-defected modules outnumbers the defected modules which lead to prediction bias towards majority class which is non-defected modules. Therefore, we searched in the literature for the best evaluation metrics that are considered reliable and work well with the existent of imbalanced data. We found that Matthews Correlation Coefficient (MCC) is the most credible metrics for imbalanced datasets [19, 41]. The MCC as shown in Eq. (1) uses binary confusion matrix values (i.e., True Positive (tp), True Negative (tn), False Positive (fp) and False Negative (fn)) and returns a value between -1 and $+1$ corresponding to the relationship between predicted

output labels and actual output labels. Any value closes to +1 means strong positive correlation, any value closes to −1 means strong negative correlation and value around zero means no correlation.

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp+fn)(tp+fp)(tn+fp)(tn+fn)}} \tag{1}$$

The second objective function is the Balanced Accuracy ($mathitBA$) as shown in Eq. (2). This measure is considered less important than MCC when data is imbalanced, but it is still useful because it computes the average of compromises between Sensitivity and Specificity. This metric is less sensitive to bias than other accuracy metrics.

$$BA = \frac{\frac{tp}{tp+fn} + \frac{tn}{tn+fp}}{2} \tag{2}$$

The Area Under Precision-Recall Curve is the third objective function ($AUPRC$). This measure is thought to be more stable and less susceptible to data imbalances than Area Under ROC [19]. $AUPRC$ is a binary response evaluation metrics statistic that is acceptable for unbalanced data and is independent of model specificity. In other fields, precision-recall curves have been recognized as useful for assessing classification performance for unbalanced binary responses; its tolerance of skewed data (e.g., many absences and few presences) makes it well suited for quantifying distribution model performance for minority cases. All objective functions are supposed to be maximized. These objective functions have been selected because, even though all of them were initially designed to show the performance of a model, they can behave in a different way as mentioned in [6, 42, 43]. This enables us to choose as many as possible good solutions that can produce tradeoff between these objective functions.

4. Multiobjective genetic algorithms

4.1. Basic concepts

Evolutionary algorithm is widely used to solve complex problems with interrelated parameters as encountered in computing. The problem of searching can be defined as follows: Given a function $f : S \to \Re$ from some set of decision vectors (S) to the set of real numbers ($\Re$), the aim is to find a solution $\vec{s_o}$ in S such that the objective function is either minimized ($f(\vec{s_o}) \leq f(\vec{s}), \forall s \in S$) or maximized ($f(\vec{s_o}) \geq f(\vec{s}), \forall s \in S$), where each solution $\vec{s}$ is defined as vector of variables in the d-dimensional space as shown in Eq. (3).

$$\vec{s} = [s_1, s_2, \ldots, s_d]^T \tag{3}$$

For single objective problem, finding the optimal solution is straight forward. But, when the problem should be optimized by multiple objective functions. We arrive at the conclusion that there is no one optimum solution, but rather a good trade-off solution that represents the best compromises between the objectives. Since there are several linked decisions that need to be optimized based on discovering trade-offs between different accuracy metrics, the challenge of tuning kNN process can be considered as a multiobjective optimization problem. The GA has been chosen for three reasons: 1) widely used among research community,

2) simple to implement and 3) it showed astonished accuracy against other evolutionary algorithms such as Particle Swarm Optimization and Simulated Annealing.

4.2. Genetic algorithm

GA is an evolutionary algorithm based on natural selection. The GA algorithm simulates natural selection, in which the fittest chromosomes are picked for reproduction in order to generate the next generation. A particular number of solutions are chosen initially to run the problem. Some of these solutions (fittest ones) are combined, resulting in new solutions including pieces (genes) from the generating pair (crossover). A new generation is defined when the components of other solutions are randomly transformed (mutation). The goal of all of these procedures is to identify the individuals who perform the best in terms of the objective function. The current best solutions are always moved to the next generation to ensure that the algorithm progresses toward improving the objective functions. A careful selection of all of these parameters, with the conventional technique being to test various combinations of their values. The size and makeup of the original population, as well as the number of generations to compute, are also essential assumptions. The last parameter of the algorithm can be securely examined by observing the evolution of the objective function over generations and terminating the algorithm when no improvement is obtained after a specific number of iterations. Finally, because the majority of the above mechanisms are influenced by random factors, it is common practice to repeat the entire operation for each fixed set of parameter values. The crossover operator randomly recombines specific parts from two selected solutions and creates a new solution (chromosome) for the new population. The mutation operator picks out a point in parent solutions and generates a new random solution to replace the previously selected solution. In contrast, the reproduction operator propagates the selected solution to the new population [44]. The process repeats until the maximum iteration set has been reached or the objective function has been met [44].

4.3. Chromosome construction

This section describes the proposed chromosome that is used with NSGAII to improve kNN model for defect prediction. Usually, the kNN model can accept two decision variables: 1) feature weights (w), and 2) number of nearest analogies (k). The first decision variable determines the importance of each feature in the training data set during distance calculation as shown in Eq. (4). The kNN classifier is defined as retrieving by similarity [42, 45]. The Euclidean distance is usually used to determine the closest observations. The value of k determines the number of nearest neighbors that will be used to make final prediction. Figure 1 illustrates the effect of k variable on the final prediction. Usually choosing small k value will result in ignoring other useful instances, whereas choosing large k value will results in potential misclassification. Therefore, the good k value is the one that can identify the actual nearest instances without degrading the accuracy.

The use of kNN is a practical application for SDP applications, given its ability to predict defects across all severity levels. The kNN method is used in addition to the other techniques, including practices such as the Euclidean distance measure to improve the accuracy of fault prediction.

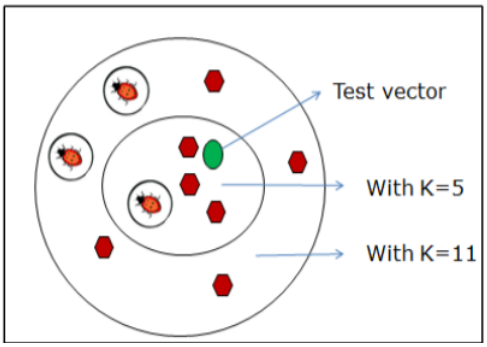


Figure 1. kNN classification example [46]

$$\text{distance} = \sqrt{\sum_{u=1}^{m} w_u \left(x_{iu} - x_{ju}\right)^2} \tag{4}$$

where m is the number of features, x_i and x_j are instances under investigation, u is the index of the feature and w_u is the weight of feature u that is identified by MOGA algorithm.

The second decision variable determines how many nearest neighbors should be retrieved from training data sets for which the final prediction will be made from them using voting technique. Each feasible solution ($\overrightarrow{s}$) in the search space is represented as a vector of two choice variables, as indicated in Eq. (5).

$$\overrightarrow{s} = \langle k, w \rangle \tag{5}$$

where k is the number of nearest neighbors which must be bounded by 1 and $n/2$ (i.e., $k \in [1, n/2]$) where n is the number of training instances, w is a numeric vector whose coordinate represent the weight in addition to the presence or absence of feature. If the any value in the w is zero, then that feature is not important. Each possible weight can take value between zero and one (i.e., $w_{ij} \in [0, 1]$) and the summation of weight values should equal to 1 as shown in Eq. (6). Finally, the dimension of w should be equivalent to number of input features in the training dataset.

$$\sum_{i=1}^{m} w_i = 1 \tag{6}$$

To illustrate that, assume the following solution vector $\overrightarrow{s} = \langle 5, 0.2, 0.4, 0.1, 0.15, 0.05, 0.1 \rangle$ are the best identified solution and assume the number of features is 6 ($m = 6$). This solution vector shows that only 5 nearest analogies should be retrieved, and the remaining values after 5 (i.e., from 0.2 to 0.1) represent the weight of each feature. It is important to note that all features should be included in distance computation because all of weights are above zero. As mentioned earlier, this solution vector is generated from NSGAII as explained in the methodology section.

4.4. Multiobjective genetic algorithm

The multiobjective optimization is defined as searching for a solution vector ($\vec{s}$) that meet d inequality constraints ($g_i(x) \geq 0, i = 1, 2, \ldots, d.$) and p equality constraints ($h_i(x) = 0, i = 1, 2, \ldots, p.$) while simultaneously optimizing a vector of M conflict objective

functions as indicated in Eq. (7). The constraints determine the feasible zone, which includes all viable options. As depicted in Figure 2, the ideal solution is reached as a trade-off between two or more competing objectives, which is known as a Pareto optimal solution. Here are some key terms to remember when it comes to multiobjective optimization:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_M(\vec{x})] \tag{7}$$

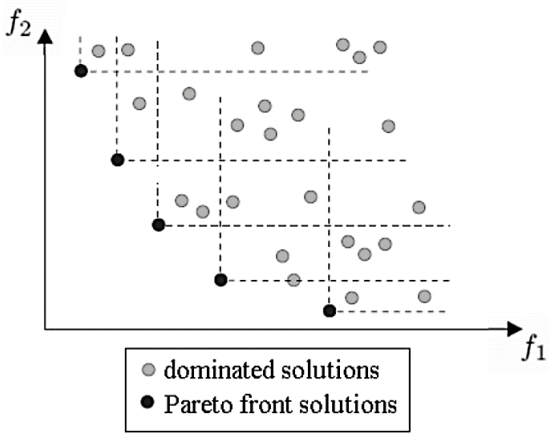


Figure 2. The Pareto front of a set of solutions in a two-objectives space (f_1 and f_2 are supposed to be minimized)

- Definition 1 (Dominance): A solutionx $\vec{x_i} \in \Re^m$ is strictly dominated by a solution $\vec{x_j} \in \Re^m$ ($\vec{x}_i \prec \vec{x}_j$ and $\vec{x}_i \neq \vec{x}_j$) iff $f_l(\vec{x}_i) \leq f_l(\vec{x}_j), \forall l \in 1, 2, \ldots, M$ and $f_l(\vec{x}_i) < f_l(\vec{x}_j), \exists l \in 1, 2, \ldots, M$.
- Definition 2 (Non-dominance): The solution $\vec{x_i} \in \Re^m$ is non-dominated solution, if there does not exist another solution $\vec{x_j} \in \Re^m$ such that $f_l(\vec{x_i}) \leq f_l(\vec{x_j}), \forall l \in 1, 2, \ldots, M$ and $i \neq j$.
- Definition 3 (Pareto optimal): We say that a solution $\vec{x}^* \in \eta \subset \Re^m$ is a Pareto optimal if it is non-dominated with respect to the feasible region (η).
- Definition 4 (Pareto optimal set): a set $\rho \subset X$ non-dominated solutions is called Pareto Optimal set which is formally defined as: $\rho = \{\vec{x} \in \eta \mid \vec{x}\}$ is a Pareto optimal set.
- Definition 5 (Pareto front): is defined as $\rho f = \{f(\vec{x}) \in \Re^m \mid \vec{x} \in \eta\}$.

In this paper, we used a modified version of the GA method called NSGAII [22], which supports Pareto-front optimization. This algorithm employs a strong elitism technique to provide a number of Pareto fronts solutions, taking non-dominance into account as illustrated in Figure 3. All non-dominated solutions make up the initial front solution. The second one has the solutions dominated by only one solution, and the fronts are generated until all solutions are classified. To maintain the diversity of solutions, another sort is performed using the crowding distance for solutions of the same front. The crowding distance defines how distant a solution's neighbors are from it, where the solutions are ranked in decreasing order. The selection operator employs both front and crowding distance sorting processes. Individuals with lower fronts are picked in the binary tournament; if the fronts are the same, the solution with the greatest crowding distance is chosen. Recombination and mutation are used to create new populations.

```
Procedure NSGA-II
  Input: N', g, f_k(X) ▷ N' members evolved g generations to solve f_k(X)
Initialize Population ℙ';
Generate random population - size N';
Evaluate Objectives Values;
Assign Rank (level) based on Pareto - sort;
Generate Child Population;
   Binary Tournament Selection;
   Recombination and Mutation;
for i = 1 to g do
   for each Parent and Child in Population do
      Assign Rank (level) based on Pareto - sort;
      Generate sets of nondominated solutions;
      Determine Crowding distance;
      Loop (inside) by adding solutions to next generation starting from
      the first front until N' individuals;
   end
   Select points on the lower front with high crowding distance;
   Create next generation;
     Binary Tournament Selection;
     Recombination and Mutation;
end
```

Figure 3. NSGAII algorithm pseudo code [22]

5. Methodology

5.1. Using the solutions produced by NSGAII in *k*NN

Figure 4 illustrates the research methodology of our study. We used repeated 10 folds cross validation to validate our model. In order to show how the NSGAII algorithm works with kNN we first start with describing the process of initialization as shown in Figure 5. Each vector represents a potential solution which is composed of two variables – k and w. Each solution is randomly initialized such that the value of k can take random integer number between 1 and $n/2$ and w can randomly take value between 0 and 1.

```
M ← 10      no. of repetition
N ← 10      no. of folds
dataset ← {CM1, MC1, AR3, PC4, KC1, KC3, Safe, Poi − 1.5, Ant − 1.3, Redktor}
foreach data ∈ dataset do
   foreach times ∈ [1, M] do
      data' ← randomize instances order
      binData ← generate N bins from data'
      foreach fold ∈ [1, N] do
         testData ← binData[fold]
         trainingData ← binData − testData
         bestSolution ← NSGAII(trainingData)
         model ← KNN(trainingData, bestSolution)
         accuracy ← evaluate(model, testData)
      end
   end
end
```

Figure 4. Research methodology

```
S ⟵ []
for i: 1 To t do
   s[t].k ⟵ Rand(1, n/2)
   foreach feature j do
      s[t].w[j] ⟵ Rand(0, 1)
   end
   S.add(s)
end
```

Figure 5. The algorithm of population generation

When NSGAII starts execution, the k and w values are updated based on crossover and mutation. Figure 3 depicts how each solution in this study is updated using Pseudo code. Recent research publications show that the solution's values frequently surpass the search space's bounds. This is most likely to take place when a solution is far away from best solution. The conventional method is to truncate the location at the exceeded boundary in this iteration and reflect the values in the boundary in the following generation such that the solution moves away. It does, however, limit the size of the solution step, limiting additional solution divergence and allowing the solution to stay near to the bounds during the search process.

In this study, we used the following configuration parameters for GA: 1) The mutation probability was 0.05, and 2) cross over operation has been performed using single point cross over.

To test the accuracy of our proposed, we also implemented a model without GA. To do this, we ran the classical kNN model and recorded the evaluation metrics to verify that the model with optimization gave us good results. We used the kNN classifier in our experiment to check the accuracy of our proposed model with and without optimization. One additional benefit of using kNN is its ability to enhance its performance as the size of the dataset increases. A larger training dataset typically provides a more comprehensive representation of the underlying data distribution. This increased coverage helps the kNN algorithm make more accurate predictions by capturing a wider range of data patterns and reducing the influence of outliers. The kNN model has also a better chance of learning the underlying patterns and relationships within the data. This improved generalization ability enables the model to make more accurate predictions on unseen or test data. The kNN model is less likely to overfit the training data. Overfitting occurs when the model becomes too specialized to the training set and fails to generalize well to new data. Increasing the training dataset size helps alleviate this issue. The kNN determines the class of a new data point based on the majority vote of its kNN. When the training dataset is larger, the decision boundaries between different classes can become more refined, leading to better classification performance.

These features ensure that software of higher complexities can use the kNN method for SDP. It is also assuring that the methodology will consider all the different parameters and produce an accurate representation of the defect probability in the system. In addition to the kNN classifier, we also tested the code using SVM, NB, and RF classifiers to compare our results and prove that the optimized model gave the best results. We chose these classifiers because they have been used widely in SDP in combination with other algorithms to evaluate results.

5.2. Datasets

The employed datasets have been obtained from the PROMISE repository [5, 12], which consisted of six imbalanced datasets from the NASA Metrics Data Program (NASA MDP), one dataset from ReLink [47], and three from the Modular Open Robotics Platform for Hackers (MORPH) [48]. These datasets are used in many studies related to SDP due to similar data points and the respective fault data, which helped us assess NSGAII-kNN performance during the testing, training, and evaluating process. Table 1 lists and describes the datasets we used in this experiment. Before implementation, we checked the datasets for missing attributes, and we applied data normalization. Before running NSGAII-kNN,

we also normalized the input features using the Min-Max scaler to avoid errors in our results.

Table 1. Datasets description

Dataset	Source	No. of attributes	No. of instances	Defective instances
CM1	NASA	37	344	12
KC1	NASA	21	2107	15
KC3	NASA	40	458	9
MC1	NASA	38	9277	1
AR3	NASA	29	63	12
PC4	NASA	37	1399	12
Safe	ReLink	26	56	39
Poi- 1.5	MORPH	20	237	59
Ant- 1.3	MORPH	20	125	16
Redktor	MORPH	20	176	15

6. Results

This section focuses on the design and implementation of NSGAII-kNN.

To answer the research questions, we first installed required libraries and packages, followed by the dataset import. We set up GA parameters manually before running the experiment. The GA was used to find the best features weight and k number that produced a minimized objective function value. Therefore, we tuned the GA parameters to a set number of maximum iterations, creating a population size. The objective function class defined how we made the population and assigned weights. Then we split our data into training and testing sets using the 10 $\times$ 10-Folds cross-validation method. For the genetic algorithm, the objective function focused on minimizing the false-negative and maximizing the true positive. Before generating the optimized function, we normalized the input features using the Min-Max scaler and then applied the kNN classifiers to output results. Next, with the aid of this optimized function class, we passed on the genetic algorithm and found out the best solution that consisted of optimal weights and k numbers. We used the 10 $\times$ 10-Folds cross-validation technique, and the iterations were repeated many times for each dataset. Finally, we recorded the output results. We used *MCC* and balanced accuracy to conclude our results. In addition, we ran the overall model on datasets CM1, KC1, KC3, MC1, PC4, AR3, Safe, Ant- 1.3, Poi- 1.5, and Redktor. Along with the optimized model setup, we ran a classical kNN model, splitting the dataset into training and testing, then normalizing the features using the Min-Max scalar before fitting the model and predicting results. We tested the accuracy of this model using the same evaluation measures as we did for the optimized model. We also set up classical SVM, NB, and RF models to compare the results with GA-kNN. The configuration parameters of these machine learning alogrithms are presented in Table 2. The layout of the code was the same as the classical kNN model. We used the same evaluation metrics to record the results.

Tables 3 and 4 summarize the results. Table 3 summarizes the *MCC* values obtained with all the models, including the SVM, NB, and RF classifier models, while table 4 shows the balanced accuracy for all models. The tables show that the NSGAII-kNN classifier performed better than the other classifier models in both *MCC* and balanced accuracy. The

Table 2. Configuration parameters of the employed machine learning algorithms

Model	Best configuration parameters
kNN	$n\_neighbors = 5$, weights='uniform', algorithm='ball_tree'
RF	n_estimator= 100, learning_rate=0.01, tree_method='hist'
NB	Kernel='Gaussian function'
SVM	Gamma='scale', kernel='rbf', epsilon=0.1

Table 3. *MCC* recorded for all models

Datasets	NSGAII kNN	Classica kNN	SVM	NB	RF
CM1	0.893	0.796	0.511	0.256	0.0963
KC1	0.974	0.950	0.967	0.433	0.379
KC3	0.326	0.198	0.056	0.278	0.0987
MC1	0.241	0.189	0.0	0.117	0.241
AR3	0.330	0.293	0.130	0.401	0.330
PC4	0.342	0.339	0.156	0.319	0.389
Safe	0.426	0.367	0.325	0.288	0.304
Poi- 1.5	0.405	0.367	0.386	0.184	0.532
Ant- 1.3	0.341	0.326	−0.009	0.234	0.0240
Redktor	0.466	0.447	0.383	0.099	0.370

Table 4. Balanced accuracy recorded for all models

Datasets	NSGAII kNN	Classical kNN	SVM	NB	RF
CM1	0.941	0.897	0.686	0.622	0.527
KC1	0.987	0.975	0.987	0.678	0.676
KC3	0.716	0.582	0.517	0.674	0.536
MC1	0.821	0.796	0.700	0.804	0.800
AR3	0.882	0.782	0.715	0.860	0.815
PC4	0.683	0.669	0.547	0.650	0.633
Safe	0.734	0.682	0.659	0.649	0.666
Poi- 1.5	0.718	0.687	0.683	0.574	0.767
Ant- 1.3	0.690	0.668	0.495	0.630	0.489
Redktor	0.771	0.723	0.719	0.510	0.722

highest *MCC* was scored with the KC1 dataset, while the highest balanced accuracy was also with the KC1 dataset. If we compare the classical kNN model with other classifiers, the kNN classifier proved to be better than the others in most of the datasets, proving to be a good classifier for SDP.

Tables 3 and 4 provide different combinations of parameters and conditions, which we used to obtain different results. NSGAII-kNN yielded the highest overall recorded *MCC* of 0.974 on the KC1 dataset, while NSGAII-kNN's lowest *MCC* value was .241 in MC1 dataset. By comparison, the classical kNN model's highest *MCC* was 0.796 in the CM1 dataset and an *MCC* of 0.189 in the MC1 dataset. Overall, the NSGAII-kNN model yielded *MCC* values ranging from 0.241 to 0.974 while the classical kNN model yielded *MCC* values ranging from 0.189 to 0.950. For balanced accuracy, again, NSGAII-kNN outperformed, yielding the highest value of 0.987. By comparison, the classical kNN model's highest value was 0.975. GA-kNN yielded balanced accuracy values between 0.650 and 0.987, while the classical kNN model yielded balanced accuracy values between 0.582 and 0.975. These results demonstrate that GA performed well on all datasets and that the NSGAII-kNN model

works better than the classical kNN model across the majority of datasets. When comparing NSGAII-kNN with the SVM, NB, and RF models, NSGAII-kNN performed better again. *MCC* and balanced accuracy remained high for GA-kNN in all the datasets and low in the SVM model, proving that NSGAII-kNN performs SDP better. The NB model also yielded poor results with *MCC* ranging from 0.099–0.433, a short-range compared to 0.241–0.974 for NSGAII-kNN. Balanced accuracy recorded with the SVM model also produced poor results with the minimum *MCC* value of 0.517 with the KC3 dataset compared to 0.650 with the optimized model. Lastly, the RF model's *MCC* range was relatively low compared to NSGAII-kNN, similar to the other classifier models. The balanced accuracy was also low, 0.489–0.815, compared to NSGAII-kNN's range of 0.650–0.975. These results demonstrate that overall, NSGAII-kNN performed better than SVM, NB, and RF.

The results obtained from these experiments used the same high-performance methods to define the accuracy of the results. Our results support previous findings that both kNN and NSGA-II are effective methodologies in SDP. Our experiments resulted in balanced accuracy and *MCC* values comparable to those found in the literature, supporting our hypothesis that using kNN with NSGAII is an effective method of SDP. Using NSGAII with kNN is a unique approach in terms of the algorithm that governs this experiment's classification methodology. Most of the literature review focused on the sole use of kNN, which has produced high-reliability results. However, our study demonstrates that the addition of GA enhances the accuracy and reliability of results and is a methodology superior to one solely reliant on kNN. Table 5 provides an analytical comparison of some of these literature reviews to our study. Many different methodologies and frameworks have been considered for application for SDP. The kNN classification method has been assessed and applied under various conditions and to other datasets, yielding results that display a high potential for application in SDP modules. The classical kNN model gave us good results when testing it as a classifier for a software defect prediction model with high accuracy values.

Finally, we revisited the proposed research questions to facilitate drawing the conclusions:

– **RQ1.** Do the use of NSGA-II and the proposed solution vector improve accuracy of kNN model for software defect prediction problem?
Answer. Yes, as we have seen from Tables 3 and 4, the proposed NSGA-II model can produce significant results over all results.

Table 5. Analytical comparison with the existing literature

Ref.	Used algorithms	Key findings in comparison to our study
[32]	Used the kNN algorithm, like our study, in addition to constructing using Euclidean distance, weighted ED, Manhattan distance, and Hausdorff distance measures.	This study achieved an accuracy of 87.2% with the same datasets as our research used. However, we used an optimized approach with GA and kNN and gained higher balanced accuracy, 94.1%, and 98.7%, with CM1 and KC1 datasets, respectively.
[49]	This study used variant-based ensemble learning and feature selection techniques.	KC1 dataset achieved an *MCC* of 0.482, very low in comparison to our study. We achieved an *MCC* of 0.974 with the KC1 dataset, proving to be a better algorithm for SDP.
[50]	The authors in this study used the kNN based probability density estimation approach using fuzzy memberships to eliminate classification errors.	The results were a balanced accuracy of 72.76% with CM1, 71.78% with KC1, and 64.27% with KC3. In comparison to these values, we obtained high-performance results that are 94.1%, 98.7%, and 65% with the CM1, KC1, and KC3 datasets, respectively.

– **RQ2.** Is there sufficient evidence that the NSGA-II algorithm can find the best k value for each module?
Answers. Yes, if we compare performance between our proposed model and classical kNN, we can reach to a conclusion that our model that uses dynamic selection of k produces better performance than classical kNN that uses static k value for all samples.

7. Threats to validity

This section aims to discuss the threats to validity encountered in the present study. The identified threats are categorized as either internal or external. External validity is further subdivided into three aspects: datasets, SDP scenarios, and evaluation measures. To ensure accurate conclusions, the utilization of an adequate number of diverse datasets that encompass a wide spectrum of features is vital. In this investigation, a comprehensive analysis was performed on 10 public datasets obtained from three software repository, which exhibit variations in module numbers and defect percentages. Moreover, the findings of this research are specific to "within-project" defect prediction scenarios, and the practical guidelines derived may not be readily applicable to other contexts. Additionally, while evaluation measures such as *MCC* and *BA* are generally robust against changes in the confusion matrix, the utilization of *MCC* and *BA* is recommended for enhanced sensitivity and informative results when dealing with imbalanced data.

The internal validity of this study is susceptible to three potential threats: the validation approach and feature selection. Although the conducted experiments employed a 10-fold cross-validation technique, it is acknowledged that leave-one-out cross-validation may yield better results by minimizing potential bias stemming from data sampling. Furthermore, feature selection algorithms were not employed in the current investigation to identify the optimal features for each dataset. Previous empirical studies have indicated that utilizing all available features often produces accuracy similar to that achieved by using the best feature subset. Consequently, the impact of the feature selection process on the final results is deemed insignificant and thus not recommended for utilization.

8. Conclusion

Predicting defects in the software modules helps developers detect faulty modules and identify the classes that might need refactoring. In this research paper, we trained the kNN classifier with a multi-optimization model called Genetic Algorithm (GA), resulting in our model, GA-kNN. Model performance was measured using *MCC* and balanced accuracy. The dataset KC1 yielded the highest *MCC* of 0.974 with $k = 3$ for GA-kNN, while the classical kNN model recorded an *MCC* value of 0.950 with the same k number. All the other datasets yielded a higher *MCC* and AUC when trained with the NSGAII-kNN model. Our experiments proved that the NSGAII-kNN model is better than the classical kNN model and can be used with various datasets. Our results demonstrated that the kNN classifier gave us good results for all the datasets proving that it is a good classifier for SDP datasets when compared to other classifiers such as SVM, NB, and RF, and thus, a good base for our model. Furthermore, NSGAII-kNN also showed superior performance when compared to other classifier models, including classical kNN because it yielded a high range of *MCC* and balanced accuracy values. In this study, we analyzed the kNN method for the purpose

of SDP, and we evaluated its performance based on three different classifiers. Our study was limited to understanding the optimization of kNN parameters using the GA. This is one possible method, although alternatives such as multilayer perceptron, neural networks and binary trees need to be analyzed for similar SDP applications on a dataset. As well, our study applied NSGAII-kNN to a limited number of datasets. Ideally, NSGAII-kNN should be applied to other varying datasets consisting of different types of data to analyze whether the results and accuracy are maintained. As well, we used three classifiers as evaluation metrics to determine the accuracy of our NSGAII-kNN model; this proves to be a limitation as numerous classifiers are available and the accuracy for SDP should be evaluated against each one of these classifiers. One final limitation of the study is that it was limited to the study of single platform algorithms and not on cross-platform systems, so it is unknown whether the proposed algorithm will provide the same results in a cross-platform environment. Future studies could investigate more developed optimization algorithms and other classification techniques for SDP. Furthermore, the NSGAII-kNN could be applied to in cross-platform applications, applications of software development, with different types and weights of datasets. Finally, future studies could evaluate the accuracy and efficiency of the proposed algorithm for use as a mainstream algorithm for SDP.

References

[1] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang et al., "Software defect prediction based on kernel PCA and weighted extreme learning machine," *Information and Software Technology*, Vol. 106, 2019, pp. 182–200.

[2] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman et al., "Local versus global lessons for defect prediction and effort estimation," *IEEE Transactions on Software Engineering*, Vol. 39, No. 6, 2013, pp. 822–834.

[3] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, Vol. 59, 2015, pp. 170–190.

[4] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, Vol. 44, No. 9, 2018, pp. 874–896.

[5] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Third International Workshop on Predictor Models in Software Engineering, PROMISE '07*, 2007.

[6] C. Tantithamthavorn, A.E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Transactions on Software Engineering*, 2018.

[7] M. Liu, L. Miao, and D. Zhang, "Two-stage cost-sensitive learning for software defect prediction," *IEEE Transactions on Reliability*, Vol. 63, No. 2, 2014, pp. 676–686.

[8] M. Singh Rawat and S. Kumar Dubey, "Software defect prediction models for quality improvement: A literature study," *IJCSI International Journal of Computer Science Issues*, Vol. 9, No. 5, 2012. [Online]. www.IJCSI.org

[9] E. Erturk and E.A. Sezer, "A comparison of some soft computing methods for software fault prediction," *Expert Systems with Applications*, Vol. 42, No. 4, 2015, pp. 1872–1879.

[10] B. Turhan, T. Menzies, A.B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, Vol. 14, No. 5, 2009, pp. 540–578. [Online]. https://link.springer.com/article/10.1007/s10664-008-9103-7

[11] I. Arora and A. Saha, "Software defect prediction: A comparison between artificial neural network and support vector machine," in *Advances in Intelligent Systems and Computing*, Vol. 562. Springer Verlag, 2018, pp. 51–61. [Online]. https://link.springer.com/chapter/10.1007/978-981-10-4603-2_6

[12] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang et al., "Defect prediction from static code features: Current results, limitations, new approaches," *Automated Software Engineering*,

Vol. 17, No. 4, 2010, pp. 375–407. [Online]. https://link.springer.com/article/10.1007/s10515-010-0069-5

[13] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: Do different classifiers find the same defects?" *Software Quality Journal*, Vol. 26, No. 2, 2018, pp. 525–552.

[14] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, Vol. 62, No. 2, 2013, pp. 434–443.

[15] X. Yang and W. Wen, "Ridge and lasso regression models for cross-version defect prediction," *IEEE Transactions on Reliability*, Vol. 67, No. 3, 2018, pp. 885–896.

[16] X. Yang, K. Tang, and X. Yao, "A learning-to-rank approach to software defect prediction," *IEEE Transactions on Reliability*, Vol. 64, No. 1, 2015, pp. 234–246.

[17] F. Wu, X.Y. Jing, Y. Sun, J. Sun, L. Huang et al., "Cross-project and within-project semisupervised software defect prediction: A unified approach," *IEEE Transactions on Reliability*, Vol. 67, No. 2, 2018, pp. 581–597.

[18] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, Vol. 46, No. 12, 2020, pp. 1267–1293.

[19] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, Vol. 45, No. 12, 2019, pp. 1253–1269.

[20] M.A. Khan, N.S. Elmitwally, S. Abbas, S. Aftab, M. Ahmad et al., "Software defect prediction using artificial neural networks: A systematic literature review," *Scientific Programming*, Vol. 2022, No. 1, 2022.

[21] M.S. Alkhasawneh, "Software defect prediction through neural network and feature selections," *Applied Computational Intelligence and Soft Computing*, Vol. 2022, No. 1, 2022, pp. 1–16.

[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, 2002, pp. 182–197.

[23] I. Maleki, A. Ghaffari, and M. Masdari, "A new approach for software cost estimation with hybrid genetic algorithm and ant colony optimization," *International Journal of Innovation and Applied Studies*, Vol. 5, No. 1, 2014, pp. 72–81. [Online]. https://www.academia.edu/download/52287242/IJIAS-13-292-35.pdf

[24] H. Alsghaier and M. Akour, "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier," *Software: Practice and Experience*, Vol. 50, No. 4, 2020, pp. 407–427.

[25] M.M. Rosli, Noor Hasimah Ibrahim Teo, Nor Shahida M. Yusop, and N. Shahriman Mohamad, "Fault prediction model for web application using genetic algorithm," in *International conference on computer and software Modeling (IPCSIT)*, 2011, pp. 71–77. [Online]. https://www.researchgate.net/profile/Marshima-Rosli/publication/264890205_Fault_Prediction_Model_for_Web_Application_Using_Genetic_Algorithm/links/55d4fcc308ae43dd17de4df4/Fault-Prediction-Model-for-Web-Application-Using-Genetic-Algorithm.pdf

[26] W. Afzal and R. Torkar, "On the application of genetic programming for software engineering predictive modeling: A systematic review," *Expert Systems with Applications*, Vol. 38, No. 9, 2011, pp. 11 984–11 997.

[27] S. Chatterjee, S. Nigam, and A. Roy, "Software fault prediction using neuro-fuzzy network and evolutionary learning approach," *Neural Computing and Applications*, Vol. 28, No. 1, 2016, pp. 1221–1231. [Online]. https://link.springer.com/article/10.1007/s00521-016-2437-y

[28] S. Goyal, "Handling class-imbalance with knn (neighbourhood) under-sampling for software defect prediction," *Artificial Intelligence Review*, Vol. 55, No. 3, 2022, pp. 2023–2064.

[29] B. Shuai, H. Li, M. Li, Q. Zhang, and C. Tang, "Software defect prediction using dynamic support vector machine," in *9th International Conference on Computational Intelligence and Security*, 2013, pp. 260–263.

[30] M.P. Sasankar and G. Sakarkar, "Cross project defect prediction using deep learning techniques," in *International Conference on Artificial Intelligence and Big Data Analytics*, 2022.

[31] K.O. Elish and M.O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, Vol. 81, No. 5, 2008, pp. 649–660.

[32] M. Hammad, A. Alqaddoumi, H. Al-Obaidy, and K. Almseidein, "Predicting software faults based on k-nearest neighbors classification," *International Journal of Computing and Digital Systems*, Vol. 8, No. 5, 2019, pp. 461–467.

[33] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "MULTI: Multi-objective effort-aware just-in-time software defect prediction," *Information and Software Technology*, Vol. 93, 2018, pp. 1–13.

[34] T.M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in *International Conference on Tools with Artificial Intelligence, ICTAI*, Vol. 1, 2010, pp. 137–144.

[35] M.A. Mabayoje, A.O. Balogun, H.A. Jibril, J.O. Atoyebi, H.A. Mojeed et al., "Parameter tuning in kNN for software defect prediction: an empirical analysis," *Jurnal Teknologi dan Sistem Komputer*, Vol. 7, No. 4, 2019, pp. 121–126.

[36] A. Iqbal, S. Aftab, I. Ullah, M. Salman Bashir, and M. Anwaar Saeed, "Modern education and computer science," *Modern Education and Computer Science*, Vol. 9, 2019, pp. 54–64. [Online]. http://www.mecs-press.org/

[37] R. Jindal, Ruchika Malhotra, and Abha Jain, "Analysis of software project reports for defect prediction using kNN," in *Proceedings of the World Congress on Engineering*, Vol. 1, 2014. [Online]. http://www.iaeng.org/publication/WCE2014/WCE2014_pp180-185.pdf

[38] D. Ulumi and D.S. Series, "Weighted knn using grey relational analysis for cross-project defect prediction," *Journal of Physics: Conference Series*, Vol. 1230, No. 1, 2019, p. 12062. [Online]. https://iopscience.iop.org/article/10.1088/1742-6596/1230/1/012062/meta

[39] R. Goyal, P. Chandra, and Y. Singh, "Suitability of kNN regression in the development of interaction based software fault prediction models," *IERI Procedia*, Vol. 6, 2014, pp. 15–21.

[40] M.A. Mabayoje, A.O. Balogun, S.M. Bello, J.O. Atoyebi, H.A. Mojeed et al., "Wrapper feature selection based heterogeneous classifiers for software defect prediction," 2019.

[41] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics*, Vol. 21, No. 1, 2020, pp. 1–13. [Online]. https://link.springer.com/articles/10.1186/s12864-019-6413-7https://link.springer.com/article/10.1186/s12864-019-6413-7

[42] D. Tomar and S. Agarwal, "Prediction of defective software modules using class imbalance learning," *Applied Computational Intelligence and Soft Computing*, Vol. 2016, 2016, pp. 1–12.

[43] A.D. Chakravarthy, S. Bonthu, Z. Chen, and Q. Zhu, "Predictive models with resampling: A comparative study of machine learning algorithms and their performances on handling imbalanced datasets," *18th IEEE International Conference on Machine Learning and Applications, ICMLA 2019*, 2019, pp. 1492–1495.

[44] P.K. Kudjo, E. Ocquaye, and W. Ametepe, "Review of genetic algorithm and application in software testing," *International Journal of Computer Applications*, Vol. 160, No. 2, 2017, pp. 1–6.

[45] S. Agarwal, D. Tomar, and Siddhant, "Prediction of software defects using twin support vector machine," in *Proceedings of the International Conference on Information Systems and Computer Networks, ISCON*. IEEE, 2014, pp. 128–132.

[46] G.D. Boetticher, "Nearest neighbor sampling for better defect prediction," *ACM SIGSOFT Software Engineering Notes*, Vol. 30, No. 4, 2005, pp. 1–6.

[47] L. Gong, S. Jiang, Q. Yu, and L. Jiang, "Unsupervised deep domain adaptation for heterogeneous defect prediction," *IEICE Transactions on Information and Systems*, Vol. E102D, No. 3, 2019, pp. 537–549.

[48] S. Hosseini, B. Turhan, and M. Mäntylä, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction," *Information and Software Technology*, Vol. 95, 2018, pp. 296–312.

[49] U. Ali, S. Aftab, A. Iqbal, Z. Nawaz, M. Salman Bashir et al., "Software defect prediction using variant based ensemble learning and feature selection techniques," *Modern Education and Computer Science*, Vol. 5, 2020, pp. 29–40. [Online]. http://www.mecs-press.org/

[50] S. Zheng, J. Gai, H. Yu, H. Zou, and S. Gao, "Software defect prediction based on fuzzy weighted extreme learning machine with relative density information," *Scientific Programming*, Vol. 2020, No. 1, 2020.

e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240104, DOI: 10.37190/e-Inf240104

Migrating a Legacy System to a Microservice Architecture

Kristian Tuusjärvi*, Jussi Kasurinen*, Sami Hyrynsalmi*
*School of Engineering Sciences, LUT University, Finland
kristian.tuusjarvi@student.lut.fi, jussi.kasurinen@lut.fi, sami.hyrynsalmi@lut.fi

Abstract

Background: In software engineering, each software product has a life cycle that at some point results in a decision being made with regard to extending its maintenance or upgrading the system to a new platform and architecture via a re-engineering or migration process. However, sometimes this decision is a non-starter; the technical dept accumulates, and platforms cease to exist, meaning that there will always be a time when extending the life support of a legacy system is no longer simply an option, and the service must be modernized.
Aim: In this paper, we focus on the migration processes, where a legacy system is updated to a microservice architecture, to understand the current state-of-the-art, applied industry practices and potential pitfalls or research gaps in the topic domain. The study aims to explore previous research to find related trends and expose gaps in the literature.
Method: We conducted a systematic mapping study on the research trends within the topic of redesign and re-engineering projects related to microservice architectures to understand what we know about microservices, what the current research trends in the area are, and if possible, what the common nominators for successful migration processes are.
Results: Our observations reveal that most microservice migration research is confined to journal articles and conference proceedings. However, a severe fragmentation in publication venues exists within the field. Furthermore, the focus of the research field is primarily on the transformation phase of the re-engineering process, with the majority of the contributions being managerial in nature, particularly of the process type. Additionally, over 50% of the research conducted is empirical in nature.
Conclusion: Based on the results, microservice migration research is maturing well; most of the research is empirical. The research field is scattered. There are notable technical, managerial, and organizational challenges and differing motivations. To better understand the motivations and challenges of the practitioners, we are going to conduct survey and interview studies within this field.

Keywords: legacy systems, microservice architecture, monolithic architecture, microservice migration

1. Introduction

In the software life cycle, each product reaches a point where a decision has to be made between extending the system maintenance, killing the software, or modernizing and migrating the legacy system into a new architecture and modern platform. In this work,

Submitted: 13 Jun. 2023; Revised: 14 Nov. 2023; Accepted: 15 Nov. 2023; Available online: 17 Nov. 2023

we conducted a systematic mapping study (SMS) [1] to study one such legacy software modernization trend – the trend of migrating monolithic legacy software systems toward microservice architecture (MSA). MSA has become popular due to recent technological advancements, such as cloud computing and containerization, which promise to allow easier service scaling, cost management, maintenance, and faster development cycles, amongst other benefits. Furthermore, the overall digitalization of the business world is forcing companies to search for more dynamic and adaptive software architectures.

1.1. Microservice technology

Microservices are small, self-sufficient processes that interact with each other using messaging protocols, such as Representational State Transfer (REST) [2–4]. MSA is a distributed cloud-native architecture that is based on service architecture (SOA) [5, 6], where developers can create, test, and deploy microservices using different development stacks and platforms [5]. Traditionally, the software has been developed as monolithic, meaning that a single executable handles all the features of a given software system [6].

Three core technologies are often utilized when using MSA. First is cloud computing; microservice based systems often run in a cloud environment where computing resources can be scaled up or down depending on the user traffic. The second is containerization; microservices are often containerized, which enables them to be deployed quickly and managed by container management software, such as Kubernetes [7]. Third, continuous integration and delivery automation processes [8] mean that the entire process, from development and quality assurance to staging and deployment, is automated to enable fast iterations and even roll-backs in case of faulty releases.

1.2. State of the industry

The research related to microservices has seen an increase during the past decade. In around 2010, the term microservice started to rise in popularity [9], with many large companies using MSA to build their software systems. For example, Sound Cloud [10], Netflix [11], and Uber [12] have adopted MSA as their service architecture. However, as the MSA model is mainly based on industry-driven needs and development, this might also correlate to the need for more literature and stricter definitions of MSA [13].

Large and complex monolithic software systems are prime candidates for MSA migration. Software systems built with MSA are less prone to accumulating complexity during their lifetimes. Software developed with distributed architecture is more self-contained than software with a monolithic architecture. It allows components to be developed and maintained separately from other parts, allowing the software system to stay robust and responsive [4]. Companies often choose to adopt MSA depending on their needs, as MSA has a reputation for having quality attributes such as availability, flexibility, maintainability, scalability, and loose coupling as built-in features [14, 15].

The motivation for this research arose from a previous study where we documented migrating from a legacy system to an MSA. During the research process, we found a lack of related research into migrating from legacy systems to MSA, which was unusual as the revision, replacement, and re-engineering work of legacy systems is not in any way an uncommon activity in the software industry. For this reason, we decided to conduct a more comprehensive literature review on migrating from legacy software to MSA.

This study aims to explore previous research to find related trends and expose gaps in the literature. In this study, we want to review the literature on migrating to MSA from legacy systems. More precisely, we want to study the research trends (publication venues and periods), the migration process phase in which migration to MSA research is focused (reverse engineering, transformation, or forward engineering), and the research contribution types to understand how MSA migration processes have been investigated in the prior works. The rest of this research paper is structured as follows: related work, methods, analysis, threats to validity, discussion, and conclusion.

2. Related work

This section discusses the research related to our topic: migrating to MSA. We will give short summaries of the related research papers, discuss their relevance to this study, and synthesize how they motivated it.

Carrasco et al. [16] conducted a literature study on microservice migration bad smells. They wanted to know what architectural and migration-related bad smells are common with MSA and how to avoid them. Their study identified nine common pitfalls as architectural smells from 58 sources, including academic and gray literature, between 2014 and 2018. The nine pitfalls are divided into five new architectural bad smells and four migration-related smells. The most common pitfalls were single-layered teams, including multiple services in one container, being greedy with containers, and simultaneously rewriting the entire system for microservices. They offer solutions for detecting and solving the pitfalls mentioned earlier. Carrasco et al.'s [16] research has an architectural focus, researching architectural pitfalls. Their study relates to ours by investigating the migration process toward MSA. However, they focus on specific architectural problems, whereas our work is more general and considers the literature and its visible trends. They also use grey literature, while we focus on academia [16].

In 2018, Knoche et al. [17] conducted a survey study on German professionals with 71 participants. They studied the primary drivers for MSA adoption, barriers to adoption, the goals of modernizing MSA, and how data consistency affects performance. They conclude that the prime drivers for modernization are scalability, maintainability, and time to market. The skills of developers and other staff were seen as the main barriers. As for goals, early adopters desired scalability from MSA, while traditional companies wanted maintainability. Performance was considered a minor issue. The authors call for similar work from other countries. They also researched migration to MSA. However, they conducted an empirical study in the form of a survey study. Furthermore, their research focuses on the motivation for migration and the barriers to adoption [17].

Velepucha et al. [18] conducted an SMS on migrating to microservices. The study included 32 primary research papers from 2012 to 2020. The research papers were only from academia. In their study, Velepucha et al. wanted to determine the types of migration proposals present in the literature and which are based on the information hiding principle. They found multiple proposals related to migrating to microservices, for example, those using DevOps, cloud computing, and performance in infrastructure. They identified that only two papers discussed migration principles, of which only one was the information hiding principle. None of the research papers proposed a software development principle to migrate from a monolithic system to MSA [18]. Our work shares some similarities with Velepucha et al. [18] as they also classified research papers based on the type of research.

However, unlike our approach, they did not use Wieringa et al.'s [19] classification. We limited our review to literature published after 2015 as we focused on the current state of the art rather than the early stages of MSA. Additionally, we analyzed various metrics related to research approaches and publication year, venue, and type.

Hassan et al. [20] conducted a large SMS based on academic and industrial literature. They analyzed 877 publications from various sources between January 2013 and April 2020. Their study had two objectives: first, to study the transition process to microservices and, second, to understand the fundamental problem of transitioning, the granularity problem of transitioning to microservices. Additionally, they classified the analyzed literature [20]. Our research shows some similarities with the SMS conducted by Hassan et al. [20] regarding MSA migrations. Like us, they also used the classification schema by Wieringa et al. [19] to classify their research. However, their research mainly focused on the issue of granularity when transitioning to microservices. Additionally, their study included grey literature, which is not the case in our research.

Auer et al. [21] conducted an interview study. They researched why companies migrate to MSA, the information metrics used, and the most helpful information metrics. Their interview study included 52 respondents from software development practitioners over five days in 2018. Based on the interviews, the authors generated an assessment framework to ease the decision-making when migrating to MSA. Their interviews with practitioners found that the most common reason for migrating to MSA was to improve maintainability. Other common reasons were independent teams, deployability, and cost, whereas modularity, complexity, fault tolerance, scalability, and reusability were less popular characteristics. The research by Auer et al. [21] relates to our study by discussing the MSA migration process. However, their study focuses on the motivation of the practitioners and the metrics they use to collect information. In contrast, our study focuses more on general information about the research field, the re-engineering phase, and the research contribution types. Moreover, their study is a survey study rather than an SMS [21].

Razzaq et al. [22] conducted an SMS study on MSA migrations. The study included 73 primary research papers from 2010 to 2021. Their goal was to evaluate the state and practice of MSA literature. They researched publication trends and venues, research focus, migration approaches and challenges, success factors post-migration, and the potential for industrial adoption. Related to the publication trends, they note that the volume of publications is progressively rising. They suggest future researchers focus on MSA in the context of the Internet of Things [22]. Razzaq et al. [22] and our study analyze publication trends, venues, and types. However, our approach differs as we delve into the publication contributions and re-engineering phase the research focuses on.

Our study on SMS migration to MSA did not have much directly comparable research available. However, we have identified the research papers that are most similar to our study in terms of the research period, method, contributions, and goals. The list of these research papers can be found in Table 1. While we found more secondary research related to MSA from different perspectives, there was no MSA-related research available in 2014, according to secondary sources cited in [23] and [24]. There has been a significant increase in MSA-related research since 2015 [23] and 2016 [24]. Waseem et al. [25] also reported a growth in MSA-related research between 2015 and 2018 [25]. Pahl et al. [23] suggest that follow-up research should be directed toward aspects such as microservices migration [23]. Furthermore, multiple authors highlight the novelty of the MSA research field [13, 20, 23, 24]. More recently (2022), Razzaq et al. [22] reported a progressively rising number of research from year to year.

Table 1. Main findings from the studies similar to ours

Author(s)	Sources	Main findings	Type	Year
Carrasco et al. [16]	58 (including grey literature)	Their study identified nine common pitfalls as architectural smells.	SMS	2018
Velepucha et al. [18]	32	They found that only two papers discussed migration principles, one of which was the information hiding principle.	SMS	2020
Hassan et al. [20]	877 (including grey literature)	They found and defined the granularity problem present in MSA migrations.	SMS	2020
Razzaq et al. [22]	73	They found that the number of publications is progressively rising.	SMS	2022

Many of the studies in our related research indicate the novelty of the research field and room for more research from different perspectives, including research within the subfield of migrating to MSA. We only found six directly related studies [16–18, 20–22], from which only four [16, 18, 20, 22] were SMSs. Our research attempts to fill this gap by summarizing the current state of research related to MSA migrations and by observing the specific re-engineering phase where the research is focused.

3. Methods

This section goes through the main phases of our research process; we followed the SMS guidelines described by Petersen et al. [1], also illustrated in Figure 1. An SMS is a research methodology that categorizes research papers and visualizes these to create a map of the researched subject [26] in the form of a conceptual map, categorization, or some other layout. An SMS is recommended as the research methodology in software engineering when the research area is still emerging and a substantial quantity of high-quality studies have yet to be completed. However, the data collection and analysis scheme is not as in-depth in similar systematic literature review models [26]. SMSs are commonly used in several domains [27], with medicine and software engineering being the most prominent areas of application [28]. The main phases of our research process are defining research questions, conducting the search, screening papers, keywording, and data extraction and mapping, as defined by Petersen et al., with all the phases having an output forwarded to the next stage; the final product is the systematic map that visualizes the results.

Based on our background work, we defined three research questions for this study. We wanted to research the period from 2015 to 2023 to capture the trends found in the microservice migration literature during that time and minimize the number of non-related

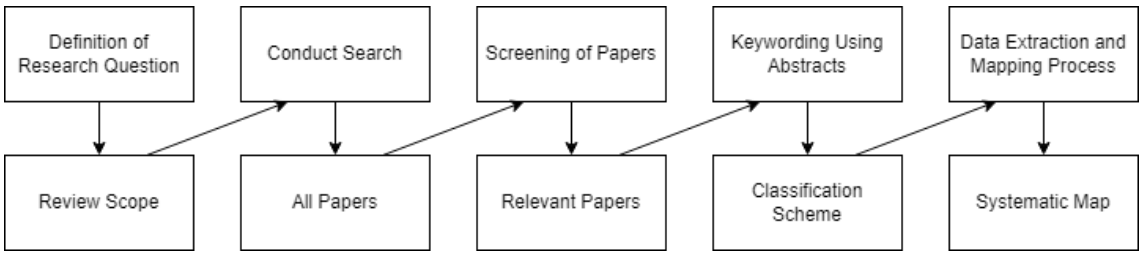


Figure 1. Systematic mapping study process as defined by Petersen et al. [1]

topics that might share similar terminology; we also wanted to analyze the research trends, the focus of research within the re-engineering process, and the contributions of the existing research work. These goals are reflected in the research questions as follows:

RQ1 – What are the research approaches implemented by the researchers?
This research question provides us with general information about the research field, which is important for analyzing the current trends in the literature.

RQ2 – What phase of the re-engineering process is addressed by the research papers?
The answer to this research question allows us to analyze which part of the re-engineering process is the most researched and where gaps in the research exist. We used the horseshoe model to define the different phases of re-engineering [29].

RQ3 – What are the contribution types of the research papers?
This question explores the concrete contributions to microservice migration research and the broader scientific community. We developed a contribution-type classification using an iterative process based on an example by Petersen et al. [1].

We used Google Scholar's research database to get research material for this SMS. We chose to use Google Scholar because it obtains research material from many different publishers and databases, such as ACM, IEEE, and Springer while having few to no limitations regarding the research domains or areas of expertise. Google Scholar can query articles with words using the following options: all, exact phrase, at least one, and without. These different options can be combined. It is possible to select where in the article the words appear: anywhere in the article or only in the title. Additionally, the author and publisher of the article can be specified, and publication dates can be indicated. We used the default search, which returned articles that included all the words from our search string, and looked for the latter anywhere in the article [30]. The only limitations we set for Google Scholar were not to include patents and the time frame. Google Scholar orders the results according to their relevance based on the full text, source, author, and number of citations [31].

The search was conducted using a search string developed by testing keywords against the database. The goal was to find a search string that yielded all the meaningful research papers that reflect the research area. The search results were evaluated manually to estimate whether they matched the research area. The evaluation was performed by the number of citations, the text's topic (related to the research topic), the author, and the source. As suggested by Petersen et al. [1], the search string reflected the research questions. The final search string was "microservices legacy software migration modernization". We conducted two searches on Google Scholar: the first in 8/2020 (from 2015 to 8/2020) and the second in 8/2023 (from 2020 to 8/2023) to update the primary research papers. The search yielded 487 initial results in 2020 and an additional 821 results in 2023, giving a total of 1308 search results. Because Google Scholar's results change over time, we have saved the original search results list to a cloud service for repeatability purposes (https://dx.doi.org/10.6084/m9.figshare.24426889).

Selection criteria were applied to the search results to filter out the unwanted results. The filtering used the inclusion and exclusion criteria shown in Tables 2 and 3, respectively. The number of citations was not considered as it would have given a less realistic view of the research area. Figure 2 shows the search process and application of the selection criteria. First, we applied our selection criteria to the initial research and removed any duplicates that could be identified. Later, during the data extraction phase, a few papers were removed as they did not fit the scope of this study.

Table 2. The inclusion criteria applied to our search results

ID	Inclusion criteria
I1	Research with more than four pages of text.
I2	Research from 2015 to 8/2020 in the first search and 2020 to 8/2023 in the second search
I3	Research in the following publication formats: books, research papers, conference papers, and journal articles.
I4	Research written in English.
I5	Research that is publicly accessible.
I6	Research that explicitly discusses the theme of this SMS (i.e., the migration of legacy software systems to MSA).

Table 3. The exclusion criteria applied to our search results

ID	Exclusion criteria
E1	Research duplicates matched with regard to the author, publication year, and title.
E2	Research that is not peer-reviewed.
E3	Research discussing microservices but not the process for migrating to MSA.

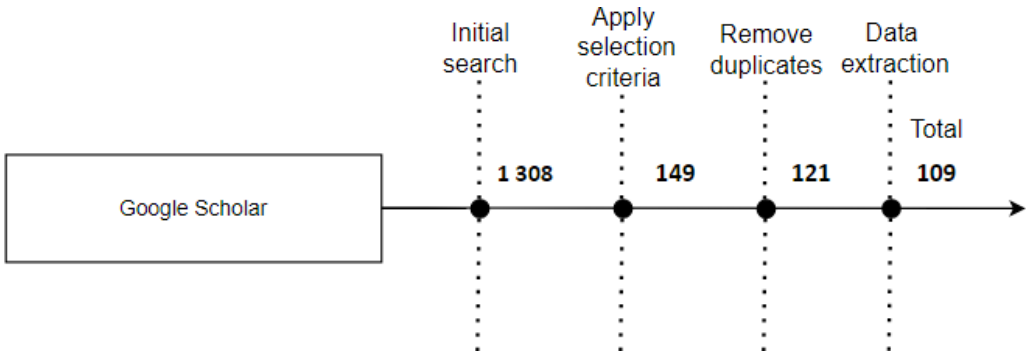


Figure 2. The process of screening the research papers

We used existing and generic classification methods to manage the data extraction process for the first two research questions. We generated a classification through the keywording process for the third research question. For the research approaches (RQ1), we used the following parameters: publication type, venue, and date; publisher; and research strategy. This research question queries general information about the research field. For the publication type, we use a simple categorization: journal, workshop paper, book, or conference proceeding and the publication venue, publisher, and date. Finally, we used the classification by Wieringa [19] to classify the research strategies, illustrated in Table 4. This categorization method is general and does not depend on any specific research field [1].

For the re-engineering phase (RQ2), we used the horseshoe model to divide the re-engineering process into reverse engineering, transformation, and forward engineering. Reverse engineering includes the acts of understanding, abstracting, and extracting a high-level model of the source system. For example, this could include software that helps understand existing systems or identifies microservice candidates. Transformation improves, restructures, and extends the previously mentioned high-level system model. For example, this could be in the form of processes or tools that help shape the new architecture. Finally, forward engineering generates a new, improved system [29]. For example, this could include guidelines and tools that help generate the new system in practice. We categorized the

Table 4. Classification of the primary papers identified, based on the principles presented in Wieringa et al. [19]

Category	Description
Validation Research	The techniques investigated are novel and have not yet been implemented. The techniques used are, for example, experiments (i.e., work done in the lab).
Evaluation Research	Techniques are implemented in practice and evaluated. This type of research shows how a technique is implemented in practice (solution implementation) and the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). This also includes identifying problems in the industry.
Solution Proposal	A solution for a problem is proposed; the solution can be either novel or a significant extension of an existing technique. A small example or a good line of argumentation shows the solution's potential benefits and applicability.
Philosophical Papers	These papers sketch a new way of looking at existing things by structuring the field in the form of a taxonomy or conceptual framework.
Opinion Papers	These papers express the opinion of somebody on whether a certain technique is good or bad or how things should be done. They do not rely on related work and research methodologies.
Experience Papers	Experience papers explain what and how something has been done in practice based on the author's personal experiences.

research papers by reading them and assigning them to one or more categories based on their topics and content, as a research paper can discuss multiple phases of the re-engineering process.

Finally, to assess the research contributions (RQ3), we classified them using an iterative keywording process by Petersen et al. [1]. We read through the studies and collected keywords and concepts representing their contributions. The context of the research was also identified. After collecting the keywords, we combined them to form a classification scheme. The classification scheme we ended up with consists of the following: process, analysis, tool, method, best practices, experience sharing, and metrics. A process is a structured approach to migrating to MSA. Analysis covers papers focusing on migration's issues and benefits and other literature. The tool assists in the migration process (i.e., software that can help with the migration process). A method provides systematic ways to achieve specific tasks within the broader migration process. Best practices are guidelines based on successful migrations. Experience sharing offers practical insights from real-world migration scenarios. The metrics can measure and evaluate different aspects of the migration process.

4. Analysis

This study aims to discover trends related to legacy software modernization, specifically migrating from legacy applications to microservices. The study was conducted as an SMS. A pool of 1308 research papers was the starting point. After the inclusion and exclusion, 109 were chosen for further analysis and categorization. In this section, we analyze the results of the categorization process.

4.1. Research areas and approaches (RQ1)

In this section, we review the results of the first research question: "(RQ1) What are the research approaches?" The first research question queried the research approaches and

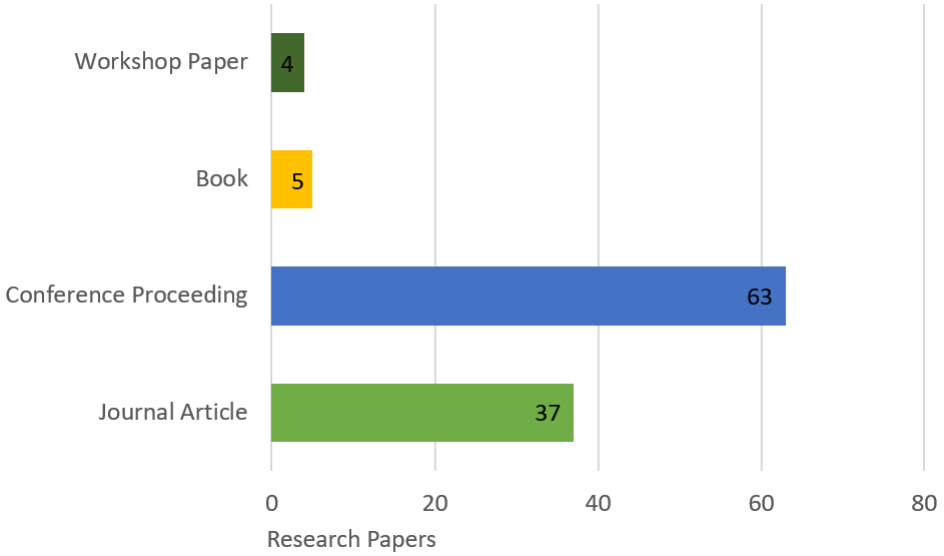


Figure 3. Publication type totals. References in Table 5

Table 5. References for the publication types

Publication type	ID
Workshop Paper	P25, P29, P36, P41
Book	P12, P15, P22, P79, P106
Conference Proceeding	P2, P3, P4, P5, P6, P7, P11, P13, P16, P17, P18, P19, P23, P26, P27, P28, P30, P31, P32, P33, P34, P35, P37, P38, P39, P40, P42, P43, P44, P46, P47, P48, P49, P50, P52, P54, P55, P57, P60, P61, P65, P66, P67, P68, P70, P72, P73, P76, P77, P80, P82, P83, P86, P88, P89, P94, P98, P99, P101, P102, P103, P104, P108
Journal Article	P1, P8, P9, P10, P14, P20, P21, P24, P45, P51, P53, P56, P58, P59, P62, P63, P64, P69, P71, P74, P75, P78, P81, P84, P85, P87, P90, P91, P92, P93, P95, P96, P97, P100, P105, P107, P109

publication information in relation to the primary research. These include the publication venue, publication time, publication type, and research type. The publication information helps us understand the current state of the research field and find any gaps that need to be filled.

We divided the publications into four groups: journal articles, conference papers, books, and workshop papers. Figure 3 shows the total numbers of the different publication types. Most research papers were either conference papers (58%) or journal articles (34%). In contrast, there is a relatively small amount of workshop papers.

Figure 4 shows the number of publications published annually from 2015 to 2023. It should be noted that 2023 has only partial data, as this study was conducted during the fall of that year. Also notable is that, in our primary research, there were no publications from 2015; the first MSA migration-related publications in our primary research material are from 2016. The lack of research is understandable since microservice technology only started to attract interest from 2010 onward [9]. We can observe a significant increase in publications from 2018, with a slight decrease in 2019 and a steady level thereafter. Regarding the publication types, we can see that conference papers and journal articles have more stable publication numbers than books and workshop papers.

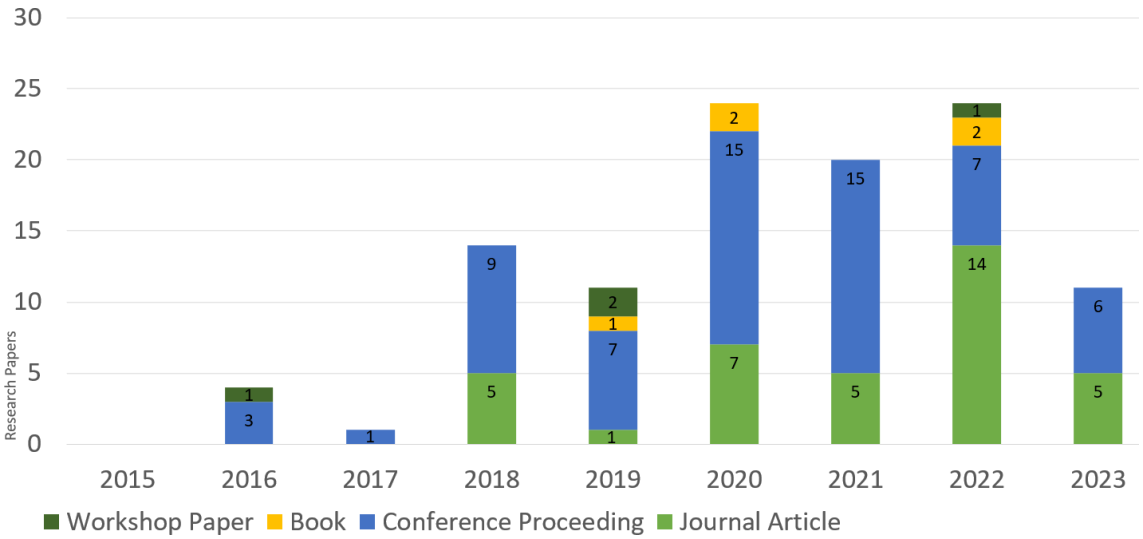


Figure 4. Publication trends from 2015 to 2023. References in Table 6

Table 6. References for the publication types per year

	Book	Conference proceeding	Journal article	Workshop paper
2016		P3, P5, P6		P36
2017		P18		
2018		P2, P13, P19, P23, P26, P30, P31, P32, P33	P1, P9, P10, P14, P21	
2019	P15	P7, P17, P27, P28, P35, P37, P48	P8	P25, P29
2020	P12, P22	P4, P11, P16, P34, P38, P39, P43, P65, P66, P67, P73, P77, P99, P103, P108	P20, P24, P62, P85, P92, P100, P107	
2021		P40, P42, P44, P46, P50, P52, P55, P60, P70, P72, P80, P83, P86, P89, P104	P58, P69, P93, P96, P105	
2022	P79, P106	P47, P57, P61, P68, P82, P94, P102	P45, P51, P53, P59, P74, P75, P78, P81, P84, P87, P91, P95, P97, P109	P41
2023		P49, P54, P76, P88, P98, P101	P56, P63, P64, P71, P90	

With regard to the various publication venues listed in Table 7, we can see severe fragmentation; only a few research papers are published through the same publication venues, meaning that 109 research papers are published through 91 different ones. The exceptions are the IEEE ICSA-C conference (4), IEEE Software Journal (3), IEEE International Conference on Software Architecture (ICSA) (3), Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (3), and International Journal of Advanced Computer Science and Applications (IJACSA) (3), all with three or more publications. The publication venues also have distinct focus areas, for example, software architecture (i.e., ICSA, ICSA-C), cloud computing (i.e., ESOCC), software maintenance (i.e., ICSME, VEM), development operations (i.e., DEVOPS), software refactoring (i.e., IWoR), data analysis (i.e., SADASC), and software engineering (i.e., APSEC, SBES, and SEAA). The rest of the publication venues are listed in Table B1 in Appendix B.

Table 7. Publication venues with more than one publication from our primary research papers. The rest of the publication venues are listed in Appendix B

Publication venue	#	ID
IEEE International Conference on Software Architecture Companion (ICSA-C)	4	P11, P34, P41, P54
IEEE Software	3	P1, P9, P10
IEEE International Conference on Software Architecture (ICSA)	3	P26, P61, P88
Euromicro Conference on Software Engineering and Advanced Applications (SEAA)	3	P38, P60, P66
International Journal of Advanced Computer Science and Applications (IJACSA)	3	P51, P63, P107
Software: Practice and Experience	2	P14, P64
On the Move to Meaningful Internet Systems: OTM Workshops	2	P16, P43
International Journal of Computer Applications (IJCA)	2	P24, P92
Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (DEVOPS)	2	P25, P29
Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS)	2	P42, P89
Empirical Software Engineering (EMSE)	2	P56, P59
International Conference on Advanced Information Systems Engineering (CAiSE)	2	P104, P108

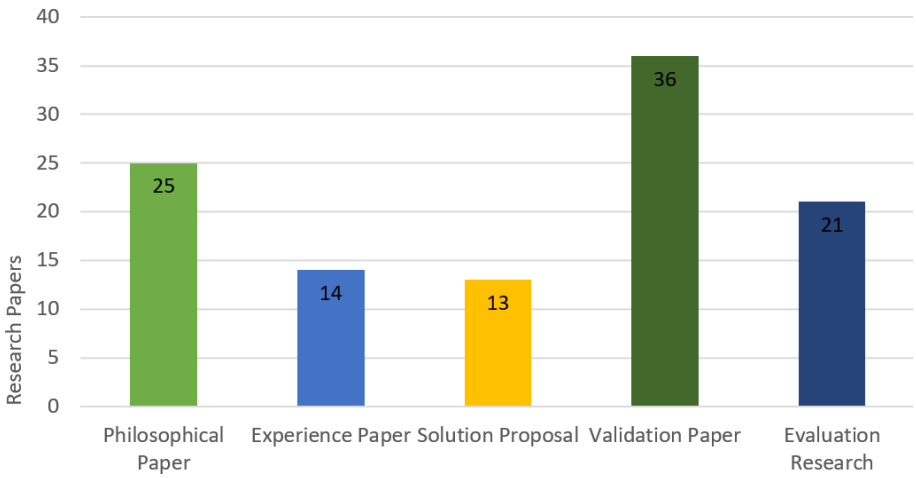


Figure 5. Research types of the papers discussing re-engineering or system migration. References in Table 8

The research types were analyzed using the classification schema by Wieringa et al. [19]. We chose this classification method because of its wide use in other systematic mapping studies (e.g., Di Francesco [24], Agilar et al. [32], Alshuqayran et al. [13]) and because it is often possible to classify a study without reading the whole paper, which saves time [1]. The classification schema consists of the following categories: validation research, evaluation research, solution proposals, philosophical papers, opinion papers, and experience papers, as listed in Table 2.

Figure 5 shows the distributions of the research types. The most used research type is validation research (33%), which investigates novel techniques in controlled environments. Philosophical papers (23%) are the second most popular research type; these papers review the research area and create taxonomies and conceptual frameworks. The third most popular research type is evaluation research (19%), meaning that many researchers are

Table 8. References for research types

Research type	ID
Philosophical paper	P7, P8, P9, P25, P26, P27, P28, P31, P35, P37, P41, P42, P44, P48, P56, P58, P64, P68, P71, P74, P79, P80, P84, P90, P100
Experience paper	P5, P10, P11, P12, P19, P30, P38, P47, P62, P66, P72, P82, P87, P103
Solution proposal	P15, P16, P17, P22, P40, P46, P53, P60, P63, P67, P81, P85, P106
Validation paper	P4, P6, P29, P32, P33, P34, P43, P49, P50, P51, P52, P55, P59, P61, P69, P70, P75, P76, P78, P83, P86, P88, P89, P91, P92, P93, P95, P96, P98, P99, P101, P102, P104, P105, P107, P108
Evaluation research	P1, P2, P3, P13, P14, P18, P20, P21, P23, P24, P36, P39, P45, P54, P57, P65, P73, P77, P94, P97, P109

testing their techniques in practice and showing the benefits and drawbacks of those techniques (evaluating their implementations). The fourth most popular research type is experience research (13%), indicating that many researchers in this field only reported their experiences. Finally, the least popular research types are solution proposals (12%) and opinion papers (0%).

The research types can be divided into empirical and non-empirical research. Non-empirical types are solution proposals, opinion papers, experience papers, and philosophical papers. Empirical study types are validation and evaluation research. Most of the studies (52%) are empirical and use verified data and observations to support research results, while only (48%) are non-empirical. Empirical studies are critical for validating theories, models, tools, and other migration-related artifacts.

4.2. Re-engineering phase (RQ2)

The second research question deals with the re-engineering phase: "(RQ2) What phase of the re-engineering process is addressed by the research papers?" To research this question, we utilized the horseshoe model, which divides the re-engineering process into three phases: reverse engineering, transformation, and forward engineering [29]. We classified the primary research papers according to the three re-engineering phases. The distribution of the re-engineering phases can be seen in Figure 6. It should be noted that a research paper can cover multiple re-engineering phases, meaning that the sum of the results is not the sum of the research papers. Almost half of the research papers (48%) focused on transformation,

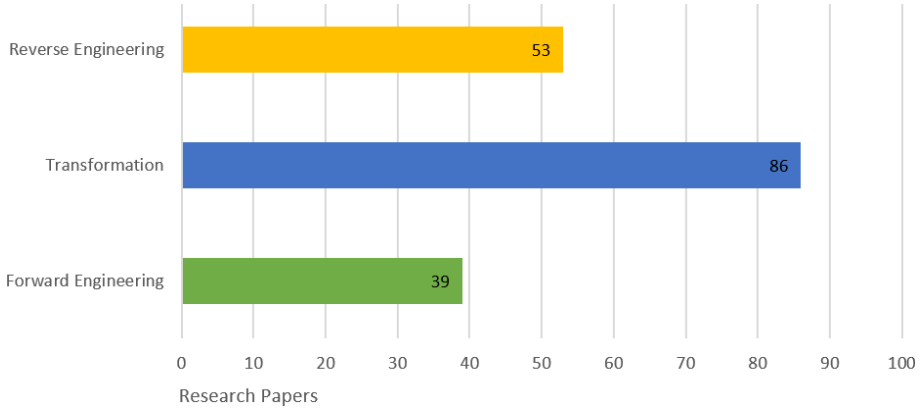


Figure 6. The re-engineering phases described as parts of the horseshoe model. References in Table 9

Table 9. References for the re-engineering phases

Re-engineering phase	ID
Reverse engineering	P1, P3, P10, P11, P13, P17, P19, P22, P23, P26, P33, P35, P36, P37, P38, P39, P40, P41, P42, P43, P44, P45, P48, P49, P52, P53, P56, P59, P60, P61, P66, P67, P68, P71, P73, P76, P78, P80, P85, P86, P89, P90, P92, P96, P98, P99, P100, P101, P102, P103, P104, P108, P109
Transformation	P2, P4, P5, P6, P7, P8, P9, P10, P11, P12, P14, P15, P16, P17, P19, P21, P22, P24, P25, P26, P27, P28, P29, P30, P31, P32, P33, P34, P35, P36, P37, P38, P39, P40, P41, P42, P43, P44, P45, P48, P49, P50, P51, P52, P54, P55, P56, P57, P58, P60, P62, P63, P65, P66, P67, P69, P70, P71, P72, P74, P75, P77, P79, P80, P81, P82, P83, P84, P85, P86, P87, P88, P90, P91, P92, P93, P94, P95, P97, P100, P103, P105, P106, P107, P108, P109
Forward engineering	P2, P5, P8, P10, P11, P12, P16, P18, P20, P22, P24, P26, P31, P37, P38, P40, P41, P42, P44, P45, P46, P47, P48, P49, P52, P56, P60, P64, P66, P67, P71, P80, P85, P86, P90, P92, P100, P103, P109

while around a third (30%) covered reverse engineering, and around a fifth (22%) focused on forward engineering, as shown in Figure 6.

4.3. Contributions to the domain (RQ3)

This section answers the third research question: "(RQ3) What are the contribution types of the research papers?" Seven different contribution categories were identified from the research papers: process, experience sharing, best practice, analysis, method, tool, and metric. Figure 7 shows the distribution of the research contributions. Nearly half (44%) of the research papers contributed to the research topic with a process, most often describing the actions that can or should be taken to accomplish the goal of migrating to MSA. After the process, the next most popular contribution type was analysis (24%), followed by tool (9%) and method (8%). Conversely, best practice (7%), experience sharing (6%), and metrics (2%) were the least common contribution types.

To better understand the contributions, they can be further categorized according to the contribution type into technical and managerial contributions. Agilar et al. [32] used this categorization method in their research for a similar purpose. A managerial contribution describes a process, method, or approach that manages the migration process. A technical

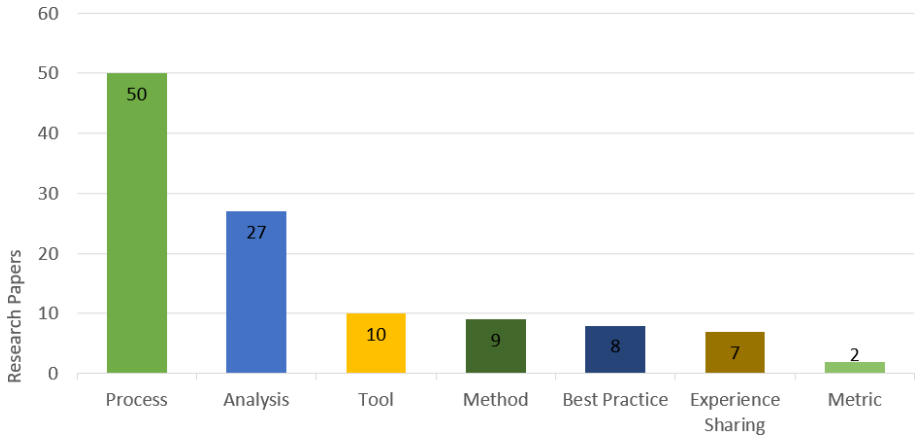


Figure 7. Distribution of the research paper contribution types toward the legacy system migration domain. References in Table 10

Table 10. References for the research types

Contribution classification	ID
Process	P1, P2, P3, P6, P10, P11, P12, P14, P15, P16, P17, P19, P21, P29, P32, P34, P36, P39, P40, P43, P45, P46, P47, P49, P50, P51, P53, P54, P59, P60, P61, P62, P65, P67, P70, P72, P73, P75, P77, P78, P81, P85, P91, P95, P96, P97, P98, P99, P106, P109
Analysis	P4, P7, P8, P18, P20, P26, P27, P28, P35, P37, P41, P42, P44, P48, P56, P57, P58, P64, P68, P71, P74, P79, P80, P82, P84, P90, P100
Tool	P13, P33, P76, P89, P93, P101, P104, P105, P107
Method	P13, P33, P76, P89, P93, P101, P104, P105, P107
Best practice	P9, P22, P25, P31, P69, P92, P100, P103
Experience sharing	P5, P30, P38, P52, P66, P87, P103
Metric	P24, P63

contribution might be a tool, a metric, or software to support migration efforts. Using this categorization method on our primary research documents yielded 90% managerial and 10% technical contributions.

4.4. Key findings

Our main findings are summarized in Table 11. While not all-inclusive, these gaps in the research and observations are worth highlighting. We can observe the publication trends, research approaches, and publication venues, as well as gain a better understanding of the MSA migration process.

Table 11. Observations regarding the research gaps related to migration processes

Source	Observation
RQ1	The year 2018 saw a significant increase in published research writings, mostly journal articles and conference papers. Overall, conference papers and journal articles dominate the publication types.
RQ1	Research into MSA migration has grown significantly between 2015 and 8/2023.
RQ1	Publication venues are scattered across different application domains and distinct topics.
RQ1	The primary research identified is split between empirical (52%) and non-empirical (48%) work.
RQ2	Primary research papers mostly focused on the transformation (48%) phase of the re-engineering process, rather than reverse engineering (30%) or forward engineering (22%)
RQ3	Managerial contributions account for 90% of all contributions, with processes being the most common type (44%).

5. Discussion

The first research question focused on the publication trends related to the primary research. Based on our results, the research on migrating legacy systems to MSA has increased from 2016 onward, as seen from Figure 4. Our primary research found no papers related to MSA migrations from 2015. However, the increasing number of publications in our primary research suggests growth, possibly supported by technological innovations (cloud platforms, containers, and DevOps), shifts in the software architectural landscape, and a move away from monolithic architecture and toward distributed architecture. Related to the growth of

MSA migration research, we observed common motivations for migration in the primary research papers: scalability (P5, P8, P12, P17, P19, P21, P25, P27, P28, P32, P42, P47, P58, P65, P71, P86, P87, P90, P103, P106, P107, P109), maintainability (P1, P2, P8, P18, P24, P25, P28, P32, P38, P42, P51, P65, P66, P77, P86, P103), time to market (P8, P30), adaptability to new technologies (P2, P25, P30), and flexibility (P42, P56, P57, P87, P90, P107). A possible reason for these motivations may be the need for more flexible software architectures in the increasingly digitalized world.

The distribution of research types in our primary research is dominated by validation research (33%), suggesting a strong emphasis on testing and validating new research artifacts in controlled settings. Research in the field of philosophy (23%) often involves the creation of taxonomies and conceptual frameworks, which provides clarity to the field and suggests a mature research area. Evaluation research, which accounts for 19% of the research conducted, serves as a bridge between theory and practice. The significant representation of evaluation research suggests that the field of research is increasingly focused on applying theoretical knowledge to practical situations and discovering the practical benefits and limitations of research techniques. Experience research accounts for only 13% of the data and is based on anecdotal evidence, lacking in testing or validation. Nevertheless, it still provides valuable real-world information. The presence of experience research indicates that practical insights and lessons learned from practitioners are still highly valued in the field. Solution proposals account for only 12% of papers in the field, focusing on validating rather than proposing new solutions.

Related to the research type, we found that a majority of it is empirical, indicating that the research field is developing and becoming more mature. This suggests that researchers are gaining a better understanding of the challenges, benefits, and nuances of migrating to MSA by validating their theories and strategies with real-world data. The abundance of validation and evaluation research also indicates that researchers are willing to test and improve the existing theories and strategies related to the migration process. Additionally, practitioners can benefit from the wealth of tried and tested tools, processes, and techniques available to them. Meanwhile, researchers have the opportunity to develop new methods by building on previous successful ones or experimenting with untested approaches.

Our primary research publications are mostly journal articles or conference proceedings. Similar results are reported in the related research by Di Francesco et al. [24] and Hassan et al. [20]. The tilt toward journals and conferences is understandable as they are more scientifically rewarding than workshop publications. Targeting more challenging publication venues is a good sign as that indicates that the research is of higher quality. We also observed fragmentation in the publication venues, as was also noted by Di Francesco et al. [24]; their study did not directly relate to ours as it focused on architecting with MSA rather than migrating to MSA; however, there is a clear parallel between their findings and ours regarding the publication venues. The fragmentation of publication venues suggests that researchers approach the MSA migration process from multiple disciplines with differing concerns. The fragmentation can also make it difficult for researchers and practitioners to navigate the literature on MSA migrations. This can make it challenging to identify influential research, gather knowledge, and ensure comprehensive coverage when conducting literature reviews. Furthermore, the fragmentation can cause redundancy as multiple researchers may work on similar research in isolation. For practitioners, this can cause obstacles when trying to access the latest best practices available, which can delay the introduction of new ones.

The second research question queried the phase of the re-engineering process that the research papers discussed. It was found that the most commonly discussed re-engineering phase was transformation, accounting for 48% of the papers, which was more popular than reverse engineering (30%) and forward engineering (22%). The difference between the transformation phase and the other re-engineering phases indicates that researchers are most interested in studying tools, workflows, and processes that transition the legacy architecture to a more modern form. This might imply that while reverse engineering and forward engineering are essential, the transformation phase is the most difficult of the three or that there are more well-established practices in the other two. From the primary research, we observed the following challenges related to the re-engineering process: the absence of suitable decomposition approaches (P28), the high level of coupling between software components (P23, P26, P27, P30, P35, P84), the lack of guidelines and best practices for migration (P7, P27, P30, P50, P67), and the identification of microservices from existing systems and boundary recognition (P3, P11, P13, P16, P17, P27, P28, P30, P32, P33, P34, P36, P39). However, many solutions are also proposed to identify microservices (P53, P55, P59, P61, P78).

The third research question aimed to identify the types of contributions made in each research paper. The results showed that the most popular contribution type was process (44%), which indicates that migration is a challenging task that requires precise and straightforward processes to guide practitioners. The analysis research type accounted for 24% of research, indicating interest in understanding migration's issues and benefits. Out of all the research articles, 9% describe tools. This indicates that the field is gradually moving toward creating software that can help with the migration process. However, further research into tooling related to the migration process toward MSA could benefit the field. Only 8% of the research articles were related to the method. This suggests less focus on refining and introducing new techniques to address specific challenges. Best practices (7%) are crucial for organizations migrating to MSA. However, their relatively low percentage suggests that the field is still consolidating these practices. As more organizations migrate, consolidating and documenting best practices will become increasingly important. Experience sharing is only responsible for 6% of contributions. Experience sharing provides valuable lessons for practitioners, given the unique challenges that each migration can present. The field could benefit from more experience-sharing contributions. Metrics account for only 2% of the contribution types. Their low percentage implies that standard metrics are yet to be developed. As the field grows, there may be an increasing demand for standardized metrics to assess the MSA migrations.

Further analysis shows that 90% of the research contributions are managerial instead of technical; Hassan et al. report similar results regarding MSA migration literature [20]. The significant difference between managerial and technical contributions might be that managerial contributions are more relevant in the migration process.

Other trends we noticed were the limitations of MSA, particularly that MSA is not a silver bullet solution for legacy migrations (P4, P5, P7, P25, P31). P4 and P7 analyzed different migration methods and concluded that there are many different methods for migration, and practitioners must choose the right one depending on their circumstances. The authors of P5 report on their experience working with an MSA migration project; they conclude that microservices are not a one-size-fits-all solution as they introduce new complexities into systems, and many factors, such as distribution complexities, should be considered before adopting this style. P31 argues that there is no single way to implement an MSA into an existing system but that practitioners should know the common pitfalls

of such processes. Additionally, it is noted in P35 that there is a lack of evidence for the benefits of mixing different migration methods.

Challenges worth researching are organizational challenges, such as the mindsets of developers during the migration process (P27), the skill sets of developers (P8, P28), and convincing management of the importance of migration (P30). Decentralizing databases is another challenge worth investigating further (P35, P26, P27, P42, P49, P68, P87, P103). In addition, our research suggests that the transactions between microservices are a challenge for practitioners to deal with, so research defining proper guidelines on migrating without performance degradation is critical (P6, P27, P38, P32).

The future of migration toward MSA can be seen through various automated tools, frameworks, and methods for migration, identification, or refactoring. These tools are highlighted in the following research papers: P75, P76, P78, P86, P88, P89, P90, P91, P95, P96, P99, P100, P101, and P108. Advanced techniques and innovations, such as reinforcement learning, are introduced in papers P81, P94, P101, and P104 to support the migration process.

Our research has identified areas with significant challenges or a lack of research, and we recommend conducting further research in these fields. Evaluation research is a significant part of primary research but could still be expanded. This is because practitioners find it challenging to adopt scientific implementations without evidence that they work in practice. There may be a lack of opportunities for evaluation research, as modernizing software from old legacy systems to MSA is still relatively rare. Additionally, companies may hesitate to invite researchers to join or find it hard to enter these large projects that continue for many years. We suggest exploring experience research and solution proposals to provide more real-world insights and challenges and address unresolved issues with innovative solutions.

Regarding the fragmentation of publication venues, we suggest several potential solutions to address this issue. These include conducting more literature reviews that provide an overview of the research field, creating centralized repositories to gather literature and increase accessibility, organizing interdisciplinary workshops and conferences to bring together researchers from different disciplines working with MSA migrations, implementing unified standards for all publication venues, promoting open access publishing to increase availability, and educating new researchers about the research field.

Related to re-engineering, much of the research is focused on the transformation phase. We suggest conducting more research on reverse and forward engineering to understand the whole migration process better. Additionally, it would be useful to investigate the challenges of re-engineering, such as the absence of suitable decomposition strategies and the handling of high coupling in software components.

It is essential to encourage more research toward developing tools and methods that can support the process of migrating. The documentation and consolidation of the best practices for MSA migration should be promoted, and experts should share experiences to capture real-world insights. Developing standardized metrics for evaluating MSA migrations is also crucial.

Promoting research that provides guidelines for choosing the appropriate migration method based on specific circumstances is important. Additionally, organizational challenges, such as those related to the developer mindset and skill set, as well as the role of management in MSA migration, should be investigated. Research on decentralizing databases and managing transactions between microservices can also be promoted. The development of automated tools, frameworks, and methods should be encouraged to simplify the MSA migration

process. Furthermore, promoting research on advanced techniques, such as reinforcement learning, can help to support the MSA migration process.

The challenges and motivations we observed from our primary research have inspired us to do more research in this field. In particular, we want to focus on the motivations and challenges related to MSA migrations. We will conduct empirical research on this topic using survey research and interviews.

6. Threats to validity

The threats to validity are classified according to the classification by Wohlin et al. [33]. They give four categories for threats to validity: conclusion, internal, external, and construct. The threats that we have identified are classified as internal and external. Internal threats are those that can affect the study results without the knowledge of the researcher. External threats limit the applicability of the results to the real world.

Regarding the internal validity threats, we implemented inclusion and exclusion criteria to enhance the exactness of the primary research further. As part of our criteria, we restricted the language to English only, which excluded 67 potential research papers from the study. Furthermore, we did not include any grey literature in our study. We do not believe that the lack of grey literature impacts the validity of our research, as peer-reviewed papers must go through strict quality gates, which improves the quality of the research papers included in this study. Another possible threat to validity is bias in selecting research papers. One researcher chose the research papers manually, which may have introduced bias in the selection process. The potential for bias was mitigated by strictly following the exclusion and inclusion criteria.

In terms of the external validity threats, the research in this paper is limited to the research discussing the migration of legacy systems to MSA. The most critical external threat to the validity of this study is that we did not do backward-forward snowballing to gather more potential primary research papers. The potentially missed primary research papers mean our study may not entirely represent the MSA migration research field. We utilized Google Scholar to search for research articles. The results are limited by publication policies, which may affect the accessibility and visibility of the results. Further limitations of Google Scholar are the inability to create complex search strings using Boolean operators and nesting. It is also only possible to search based on the title or full text; it cannot define proximity to the searched words; it cannot use complex dates, only date ranges. In addition, the subject area is broad because there are no predefined sections for each subject matter.

7. Conclusion

Interest in MSA has seen an increase during the past decade. The interest is fuelled by technological advancements, such as cloud computing, automation (DevOps), and containerization, as well as by the overall trend of digitalization and the way software is consumed through the internet via browsers and mobile devices. Therefore, in this study, we have studied the literature related to migrating from legacy systems to MSA.

We identified a pool of 1308 research publications and narrowed it down to 109 primary sources discussing migrating from a legacy system to an MSA-based system. Our first research question focused on the primary research and, more specifically, looked into the

research strategy, publication year, and publication venue. The second research question covered the specific area of research related to migrating from legacy systems to MSA. Lastly, the third research question investigated the contributions of the research papers.

From the primary document analysis, we identified the following observations. Legacy system to MSA migration research has increased from 2016 and reached a stable level from 2018 onward. The amount of evaluation research suggests that the research field is maturing. Another indication is that researchers in this field mostly target challenging publication venues (conferences and journals). Related to the publication venues, we also observed that there is major fragmentation in the publication forums, which suggests that researchers approach the MSA migration process from multiple disciplines with differing goals. This can make the research harder to find as it is scattered across many different publication forums. Regarding the focus of the research (estimated with the horseshoe model), we found that more studies focused on the transformation phase compared to reverse engineering or forward engineering. Finally, the most common research contribution type was a process.

The other trends identified from the literature include migration motivations: scalability, maintainability, time to market, and adaptability to new technologies. We also note the challenges observed in the research: that MSA is not a silver bullet solution for legacy migrations, the decomposition of existing systems and identifying microservice candidates from existing legacy systems, organizational challenges, decentralizing databases, the migration of databases, and performance degradation during migration.

The MSA migration research is mature based on the publication venues and research types utilized. The research field is scattered across many publication venues. There are notable technical, managerial, and organizational challenges and differing motivations. We have included our recommendations for future research in the discussion section. We will continue conducting research in this field with survey and interview studies to study industry practitioner's challenges and motivations to understand the current problems and pitfalls that affect the migration processes and the reverse engineering and re-engineering tasks required.

Information about funding/support sources

We received no funding for this research.

References

[1] 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)*, 2008, pp. 1–10.

[2] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Software*, Vol. 33, No. 3, 2016, pp. 42–52.

[3] M. Garriga, "Towards a taxonomy of microservices architectures," in *Software Engineering and Formal Methods. SEFM 2017*, Lecture Notes in Computer Science, A. Cerone and M. Roveri, Eds., Vol. 10729. Springer, 2018, pp. 203–218.

[4] M. Richards, *Microservices vs. service-oriented architecture*. O'Reilly Media, 2016.

[5] X. Larrucea, I. Santamaria, R. Colomo-Palacios, and C. Ebert, "Microservices," *IEEE Software*, Vol. 35, No. 3, 2018, pp. 96–100.

[6] N. Dragoni, S. Dustdar, S.T. Larsen, and M. Mazzara, "Microservices: Migration of a mission critical system," *arXiv preprint arXiv:1704.04173*, 2017.

[7] P. Jamshidi, C. Pahl, N.C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, Vol. 35, No. 3, 2018, pp. 24–35.
[8] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Migrating to cloud-native architectures using microservices: An experience report," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2015, pp. 201–215.
[9] M. Fowler, "Microservices guide," martinfowler.com, Tech. Rep., 2014.
[10] P. Calcado, "Building products at SoundCloud – Part II: Breaking the monolith," SoundCloud, Tech. Rep., 2014. [Online]. https://developers.soundcloud.com/blog/building-products-at-soundcloud-part-2-breaking-the-monolith
[11] C. Bampis, C. Chen, A. Moorthy, and Z. Li, "Netflix video quality at scale with cosmos microservices." Medium, Tech. Rep., 2021. [Online]. https://netflixtechblog.com/netflix-video-quality-at-scale-with-cosmos-microservices-552be631c113
[12] E. Haddad, "Service-oriented architecture: Scaling the Uber engineering codebase as we grow." Uber, Tech. Rep., 2015. [Online]. https://eng.uber.com/service-oriented-architecture/
[13] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture." in *9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Vol. 9. IEEE, 2016, pp. 44–51.
[14] D.S. Linthicum, "Practical use of microservices in moving workloads to the cloud," *IEEE Cloud Computing*, Vol. 3, No. 5, 2016, pp. 6–9.
[15] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce," in *International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017.
[16] A. Carrasco, B.V. Bladel, and S. Demeyer, "Migrating towards microservices: Migration and architecture smells," in *Proceedings of the 2nd International Workshop on Refactoring*. ACM, 2018, pp. 1–6.
[17] H. Knoche and W. Hasselbring, "Drivers and barriers for microservice adoption – A survey among professionals in Germany. Enterprise modelling and information systems architectures," *IEEE Software*, Vol. 14, 2019, pp. 1–35.
[18] V. Velepucha, P. Flores, J. Torres, M. Botto-Tobar, J. León-Acurio et al., "Migration of monolithic applications towards microservices under the vision of the information hiding principle: A systematic mapping study," in *Advances in Emerging Trends and Technologies*. Springer, 2020, pp. 90–100.
[19] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requirements engineering*, Vol. 11, No. 1, 2006, pp. 102–107.
[20] S. Hassan, R. Bahsoon, and R. Kazman, "Microservice transition and its granularity problem: A systematic mapping study," *Software: Practice and Experience*, Vol. 50, 2020, pp. 1651–1681.
[21] H. Knoche and W. Hasselbring, "From monolithic systems to microservices: An assessment framework," *Information and Software Technology*, Vol. 137, 2021.
[22] A. Razzaq and S.A.K. Ghayyur, "A systematic mapping study: The new age of software architecture from monolithic to microservice architecture – awareness and challenges," *Computer Applications in Engineering Education*, Vol. 31, No. 2, 2023, pp. 421–451.
[23] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study," in *6th International Conference on Cloud Computing and Services Science*, Vol. 1. ACM, 2016, pp. 137–146.
[24] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *Journal of Systems and Software*, Vol. 150, 2019, pp. 77–97.
[25] M. Waseem, P. Liang, and M. Shahin, "A systematic mapping study on microservices architecture in DevOps," *Journal of Systems and Software*, Vol. 170, 2020.
[26] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," EBSE, Tech. Rep., 2007.
[27] E. Vanhala, J. Kasurinen, A. Knutas, and A. Herala, "The application domains of systematic mapping studies: A mapping study of the first decade of practice with the method," *IEEE Access*, Vol. 10, 2022, pp. 37 924–37 937.

[28] J. Bailey, D. Budgen, M. Turner, B. Kitchenham, P. Brereton et al., "Evidence relating to object-oriented software design: A survey," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE, 2007, pp. 482–484.
[29] R. Kazman, S. Woods, and C. S.J, "Requirements for integrating software architecture and reengineering models: CORUM II," in *Proceedings Fifth Working Conference on Reverse Engineering*. IEEE, 1998.
[30] M. Bangert, *Research Guides: Google Scholar: Advanced Searching*. [Online]. https://semo.libguides.com/google-scholar/advanced-searching
[31] S.H.S. University, *What is Google Scholar and how do I use it?*, 2022. [Online]. https://www.shsu.edu/research/guides/tutorials/googlescholar/index.html
[32] E. de Vargas Agilar, R.B. de Almeida, and E.D. Canedo, "A systematic mapping study on legacy system modernization," *SEKE*, 2016, pp. 345–350.
[33] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell et al., *Experimentation in Software Engineering*. Springer. [Online]. http://link.springer.com/10.1007/978-3-642-29044-2

Appendix A. Primary studies

Table A1. Primary research papers

ID	Title	Author	Year
P1	Using Microservices for Legacy Software Modernization	Holger Knoche, Wilhelm Hasselbring	2018
P2	On the Modernization of ExplorViz towards a Microservice Architecture	Christian Zirkelbach, Alexander Krause, Wilhelm Hasselbring	2018
P3	Towards the Understanding and Evolution of Monolithic Applications as Microservices	Daniel Escobar, Diana Cárdenas, Rolando Amarillo, Eddie Castro, Kelly Garcés, Carlos Parra, Rubby Casallas	2016
P4	Analysis of Legacy Monolithic Software Decomposition into Microservices	Justas Kazanavičius, Dalius Mazeika	2020
P5	Migrating to cloud-native architectures using microservices: An experience report	Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Antonio Celesti, Philipp Leitner	2016
P6	Sustaining Runtime Performance while Incrementally Modernizing Transactional Monolithic Software towards Microservices	Holger Knoche	2016
P7	Migrating Legacy Software to Microservices Architecture	Justas Kazanavičius, Dalius Mažeika	2019
P8	Drivers and Barriers for Microservice Adoption – A Survey among Professionals in Germany	Holger Knoche, Wilhelm Hasselbring	2019
P9	Migrating enterprise legacy source code to microservices: On multitenancy, statefulness, and data consistency	Andrei Furda, Colin Fidge, Olaf Zimmermann, Wayne Kelly, Alistair Barros	2018
P10	Microservices	Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, Christof Ebert	2018
P11	Microservice Decomposition via Static and Dynamic Analysis of the Monolith	Alexander Krause, Christian Zirkelbach, Wilhelm Hasselbring, Stephan Lenga, Dan Kröger	2020
P12	Principles of the Newdimensions Software Creation for a Control Centre of the Future: Cloud Computing and Software Architecture	Rúben Araújo, Joaquim Nunes, Afonso Fernandes, Rolando Martins	2020
P13	Extracting Candidates of Microservices from Monolithic Application Code	Manabu Kamimura, Keisuke Yano, Tomomi Hatano, Akihiko Matsuo	2018
P14	Microservices migration patterns	Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Damian A. Tamburri, Theo Lynn	2018
P15	Migrating to Microservices	Alexis Henry, Youssef Ridene, Antonio Bucchiarone, Nicola Dragoni, Schahram Dustdar, Patricia Lago, Manuel Mazzara, Victor Rivera, Andrey Sadovykh	2019
P16	Translating a Legacy Stack to Microservices Using a Modernization Facade with Performance Optimization for Container Deployments	Prabal Mahanta, Suchin Chouta, Christophe Debruyne, Hervé Panetto, Wided Guédria, Peter Bollen, Ioana Ciuciu, George Karabatis, Robert Meersman	2020
P17	From Monolith to Cloud Architecture Using Semi-automated Microservices Modernization	Salvatore Augusto Maisto, Beniamino Di Martino, Stefania Nacchia, Leonard Barolli, Peter Hellinckx, Juggapong Natwichai	2019

Table A1 continued

ID	Title	Author	Year
P18	Using Microservices and Software Product Line Engineering to Support Reuse of Evolving Multi-tenant SaaS	Leonardo P. Tizzei, Marcelo Nery, Vinícius C.V.B. Segura, Renato F.G. Cerqueira	2017
P19	Cracking the Monolith: Challenges in Data managementing to Cloud Native Architectures	Mishra Mayank, Kunde Shruti, Nambiar Manoj	2018
P20	Does Migrate a Monolithic System to Microservices Decreases the Technical Debt?	Valentina Lenarduzzi, Francesco Lomio, Nyyti Saarimäki, Davide Taibi	2020
P21	Microservices: Migration of a Mission Critical System	Manuel Mazzara, Nicola Dragoni, Antonio Bucchiarone, Alberto Giaretta, Stephan T. Larsen, Schahram Dustdar	2018
P22	Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example	Alan Megargel, Venky Shankararaman, David K. Walker, Muthu Ramachandran, Zaigham Mahmood	2020
P23	Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems	Adambarage Anuruddha Chathuranga De Alwis, Alistair Barros, Artem Polyvyanyy, Colin Fidge, Claus Pahl, Maja Vukovic, Jianwei Yin, Qi Yu	2018
P24	A Decoupled Health Software Architecture using Microservices and OpenEHR Archetypes	Marcio Silva, André Araújo, Paulo Caetano da Silva	2020
P25	From Monolith to Microservices: A Classification of Refactoring Approaches	Jonas Fritzsch, Justus Bogner, Alfred Zimmermann, Stefan Wagner	2019
P26	Migrating Towards Microservice Architectures: An Industrial Survey	Paolo Di Francesco, Patricia Lago, Ivano Malavolta	2018
P27	Strategies Reported in the Literature to Migrate to Microservices Based Architecture	Heleno Cardoso da Silva Filho, Glauco de Figueiredo Carneiro, Shahram Latifi	2019
P28	Microservices Migration in Industry: Intentions, Strategies, and Challenges	Jonas Fritzsch, Justus Bogner, Stefan Wagner, Alfred Zimmermann	2019
P29	A Model-Driven Approach Towards Automatic Migration to Microservices	Antonio Bucchiarone, Kemal Soysal, Claudio Guidi, Jean-Michel Bruel, Manuel Mazzara, Bertrand Meyer	2019
P30	An Experience Report on the Adoption of Microservices in Three Brazilian Government Institutions	Welder Luz, Everton Agilar, Marcos César de Oliveira, Carlos Eduardo R. de Melo, Gustavo Pinto, Rodrigo Bonifácio	2018
P31	Migrating towards microservices: Migration and architecture smells	Andrés Carrasco, Brent van Bladel, Serge Demeyer	2018
P32	Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic	Adambarage Anuruddha Chathuranga De Alwis, Alistair Barros, Colin Fidge, Artem Polyvyanyy, Hervé Panetto, Christophe Debruyne, Henderik A. Proper, Claudio Agostino Ardagna, Dumitru Roman, Robert Meersman	2018
P33	Migrating Web Applications from Monolithic Structure to Microservices Architecture	Zhongshan Ren, Wei Wang, Guoquan Wu, Chushu Gao, Wei Chen, Jun Wei, Tao Huang	2018
P34	Towards Identifying Microservice Candidates from Business Rules Implemented in Stored Procedures	Marx Haron Gomes Barbosa, Paulo Henrique M. Maia	2020
P35	Migrating from monolithic architecture to microservices: A Rapid Review	Francisco Ponce, Gastón Márquez, Hernán Astudillo	2019
P36	Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems	Alessandra Levcovitz, Ricardo Terra, Marco Tulio Valente	2016

Table A1 continued

ID	Title	Author	Year
P37	Migration to Microservices: Barriers and Solutions	Javad Ghofrani, Arezoo Bozorgmehr	2019
P38	From a Monolithic Big Data System to a Microservices Event-Driven Architecture	Rodrigo Laigner, Marcos Kalinowski, Pedro Diniz, Leonardo Barros, Carlos Cassino, Melissa Lemos, Darlan Arruda, Sergio Lifschitz, Yongluan Zhou	2020
P39	Automatic Microservices Identification from a Set of Business Processes	Mohamed Daoud, Asmae El Mezouari, Noura Faci, Djamal Benslimane, Zakaria Maamar, Aziz El Fazziki	2020
P40	Modernizing legacy systems with microservices: A roadmap	Daniele Wolfart, Wesley K.G. Assunção, Ivonei F. da Silva, Diogo C.P. Domingos, Ederson Schmeing, Guilherme L. Donin Villaca, Diogo do N. Paza	2021
P41	A Systematic Literature Review on Migration to Microservices: A Quality Attributes perspective	Roberta Capuano, Henry Muccini	2022
P42	Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices	Thelma Colanzi, Aline Amaral, Wesley Assunção, Arthur Zavadski, Douglas Tanno, Alessandro Garcia, Carlos Lucena	2021
P43	Translating a legacy stack to microservices using a modernization facade with performance optimization for container deployments	Prabal Mahanta, Suchin Chouta	2020
P44	Monoliths to microservices-migration problems and challenges: A SMS	Victor Velepucha, Pamela Flores	2021
P45	SPReaD: service-oriented process for reengineering and DevOps: Developing microservices for a Brazilian state department of taxation	Carlos Eduardo da Silva, Yan de Lima Justino, Eiji Adachi	2022
P46	Migration of monoliths through the synthesis of microservices using combinatorial optimization	Gianluca Filippone, Marco Autili, Fabrizio Rossi, Massimo Tivoli	2021
P47	The Adoption of Microservices Architecture as a Natural Consequence of Legacy System Migration at Police Intelligence Department	Murilo Góes de Almeida, Edna Dias Canedo	2022
P48	Migration of monolithic applications towards microservices under the vision of the information hiding principle: A systematic mapping study	Victor Velepucha, Pamela Flores, Jenny Torres	2019
P49	An Approach to Migrate from Legacy Monolithic Application into Microservice Architecture	Justas Kazanavičius, Dalius Mažeika	2023
P50	A multi-criteria strategy for redesigning legacy features as microservices: An industrial case study	Wesley K.G. Assunção, Thelma Elita Colanzi, Luiz Carvalho, Juliana Alves Pereira, Alessandro Garcia, Maria Julia de Lima, Carlos Lucena	2021
P51	From Monolith to Microservices: A Semi-Automated Approach for Legacy to Modern Architecture Transition using Static Analysis	Mohd Hafeez Osman, Cheikh Saadbouh, Khaironi Yatim Sharif, Novia Admodisastro	2022
P52	Design Patterns and Microservices for Reengineering of Legacy Web Applications	V. Dattatreya, K.V. Chalapati Rao, M. Raghava	2021
P53	Improving microservices extraction using evolutionary search	Khaled Sellami, Ali Ouni, Mohamed Aymen Saied, Salah Bouktif, Mohamed Wiem Mkaouer	2022

Table A1 continued

ID	Title	Author	Year
P54	The Quality-Driven Refactoring Approach in BIM Italia	Roberta Capuano, Fabio Vaccaro	2023
P55	Applying Microservice Refactoring to Object-oriented Legacy System	Junfeng Zhao, Ke Zhao	2021
P56	An empirical study of the systemic and technical migration towards microservices	Hamdy Michael Ayas, Philipp Leitner, Regina Hebig	2023
P57	Towards a Multi-Tenant Microservice Architecture: An Industrial Experience	Cesar Batista, Bruno Proença, Everton Cavalcante, Thais Batista, Felipe Morais, Henrique Medeiros	2022
P58	Review of methods for migrating software systems to microservices architecture	Aleksandra Stojkov, Zeljko Stojanov	2021
P59	Analysis of a many-objective optimization approach for identifying microservices from legacy systems	Wesley K.G. Assunção, Thelma Elita Colanzi, Luiz Carvalho, Alessandro Garcia, Juliana Alves Pereira, Maria Julia de Lima, Carlos Lucena	2022
P60	Migrating monoliths to microservices-based customizable multi-tenant cloud-native apps	Sindre Grønstøl Haugeland, Phu H. Nguyen, Hui Song, Franck Chauvel	2021
P61	Leveraging the layered architecture for microservice recovery	Pascal Zaragoza, Abdelhak-Djamel Seriai, Abderrahmane Seriai, Anas Shatnawi, Mustapha Derras	2022
P62	The collaborative modularization and reengineering approach CORAL for open source research software	Christian Zirkelbach, Alexander Krause, Wilhelm Hasselbring	2020
P63	From Monolith to Microservice: Measuring Architecture Maintainability	Muhammad Hafiz Hasan, Mohd. Hafeez Osman, Novia Indriaty Admodisastro, Muhamad Sufri Muhammad	2023
P64	Adopting microservices and DevOps in the cyber-physical systems domain: A rapid review and case study	Jonas Fritzsch, Justus Bogner, Markus Haug, Ana Cristina Franco da Silva, Carolin Rubner, Matthias Saft, Horst Sauer, Stefan Wagner	2023
P65	Microservice migration using strangler fig pattern: A case study on the green button system	Chia-Yu Li, Shang-Pin Ma, Tsung-Wen Lu	2020
P66	From a monolithic big data system to a microservices event-driven architecture	Rodrigo Laigner, Marcos Kalinowski, Pedro Diniz, Leonardo Barros, Carlos Cassino, Melissa Lemos, Darlan Arruda, Sérgio Lifschitz, Yongluan Zhou	2020
P67	Towards a process for migrating legacy systems into microservice architectural style	Daniele Wolfart, Ederson Schmeing, Gustavo Geraldino, Guilherme Villaca, Diogo Paza, Diogo Paganini, Wesley K.G. Assunção, Ivonei F. da Silva, Victor F.A. Santander	2020
P68	Using Database Schemas of Legacy Applications for Microservices Identification: A Mapping Study	Antonios Mparmpoutis, George Kakarontzas	2022
P69	Patterns for Migration of SOA Based Applications to Microservices Architecture.	Vinay Raj, Ravichandra Sadam	2021
P70	A Hot Decomposition Procedure: Operational Monolith System to Microservices	Nikolay Ivanov, Antoniya Tasheva	2021
P71	A systematic mapping study: The new age of software architecture from monolithic to microservice architecture–awareness and challenges	Abdul Razzaq, Shahbaz A.K. Ghayyur	2023

Table A1 continued

ID	Title	Author	Year
P72	From Monolithic to Microservice Architecture: The Case of Extensible and Domain-specific IDEs	Romain Belafia, Pierre Jeanjean, Olivier Barais, Gurvan Le Guernic, Benoit Combemale	2021
P73	On the performance and adoption of search-based microservice identification with tomicroservices	Luiz Carvalho, Alessandro Garcia, Thelma Elita Colanzi, Wesley K.G. Assunção, Juliana Alves Pereira, Baldoino Fonseca, Márcio Ribeiro, Maria Julia de Lima, Carlos Lucena	2020
P74	Transformation of Monolithic Applications towards Microservices	Zaigham Mushtaq, Najia Saher, Faisal Shazad, Sana Iqbal, Anam Qasim, Imran Imran	2022
P75	An Approach to Migrate a Monolith Database into Multi-Model Polyglot Persistence Based on Microservice Architecture: A Case Study for Mainframe ...	Justas Kazanavičius, Dalius Mažeika, Diana Kalibatienė	2022
P76	Code vectorization and sequence of accesses strategies for monolith microservices identification	Vasco Faria, António Rito Silva, Irene Garrigós, Juan Manuel Murillo Rodríguez, Manuel Wimmer	2023
P77	Microservice Decompositon: A Case Study of a Large Industrial Software Migration in the Automotive Industry.	Heimo Stranner, Stefan Strobl, Mario Bernhart, Thomas Grechenig	2020
P78	Expert system for automatic microservices identification using API similarity graph	Xiaoxiao Sun, Salamat Boranbaev, Shicong Han, Huanqiang Wang, Dongjin Yu	2022
P79	Re-engineering Legacy Systems as Microservices: An Industrial Survey of Criteria to Deal with Modularity and Variability of Features	Luiz Carvalho, Alessandro Garcia, Wesley K.G. Assunção, Thelma Elita Colanzi, Rodrigo Bonifácio, Leonardo P. Tizzei, Rafael de Mello, Renato Cerqueira, Márcio Ribeiro, and Carlos Lucena	2022
P80	The migration journey towards microservices	Hamdy Michael Ayas, Philipp Leitner, Regina Hebig	2021
P81	Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning	MohammadHadi Dehghani, Shekoufeh Kolahdouz-Rahimi, Massimo Tisi, Dalila Tamzalit	2022
P82	Migration from monolithic to microservice architecture: Research of impacts on agility	Josef Doležal, Alena Buchalcevová	2022
P83	Mono2micro: A practical and effective tool for decomposing monolithic java applications to microservices	Anup K. Kalia, Jin Xiao, Rahul Krishna, Saurabh Sinha, Maja Vukovic, Debasish Banerjee	2021
P84	Accumulation and prioritization of architectural debt in three companies migrating to microservices	Saulo Soares De Toledo, Antonio Martini, Phu H. Nguyen, Dag I.K. Sjøberg	2022
P85	How to transition incrementally to microservice architecture	Karoly Bozan, Kalle Lyytinen, Gregory M. Rose	2020
P86	Materializing Microservice-oriented Architecture from Monolithic Object-oriented Source Code	Pascal Zaragoza, Abdelhak-Djamel Seriai, Abderrahmane Seriai, Anas Shatnawi, Hinde-Lilia Bouziane, Mustapha Derras	2021
P87	Developing a Microservices Integration Layer for Next-Generation Rail Operations Centers	Andrei Furda, Lionel van den Berg, Graeme Reid, Giancarlo Camera, Matteo Pinasco	2022

Table A1 continued

ID	Title	Author	Year
P88	From monolithic to microservice architecture: An automated approach based on graph clustering and combinatorial optimization	Gianluca Filippone, Nadeem Qaisar Mehmood, Marco Autili, Fabrizio Rossi, Massimo Tivoli	2023
P89	Microservice decomposition and evaluation using dependency graph and silhouette coefficient	Ana Santos, Hugo Paula	2021
P90	Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review	Yalemisew Abgaz, Andrew McCarren, Peter Elger, David Solan, Neil Lapuz, Marin Bivol, Glenn Jackson, Murat Yilmaz, Jim Buckley, Paul Clarke	2023
P91	Migrating Monoliths to Microservices Integrating Robotic Process Automation into the Migration Approach	Burkhard Bamberger, Bastian Körber	2022
P92	Mind Overflow: A Process Proposal for Decomposing Monolithic Applications in Microservices	Tcharles Pereira, Kleinner Farias	2020
P93	Benchmarks and performance metrics for assessing the migration to microservice-based architectures.	Nichlas Bjørndal, Antonio Bucchiarone, Manuel Mazzara, Nicola Dragoni, Schahram Dustdar	2021
P94	CARGO: AI-guided dependency analysis for migrating monolithic applications to microservices architecture	Vikram Nitin, Shubhi Asthana, Baishakhi Ray, Rahul Krishna	2022
P95	From Legacy to Microservices: A Type-based Approach for Microservices Identification using ML and Semantic Analysis	Imen Trabelsi, Manel Abdellatif, Abdalgader Abubaker, Naouel Moha, Sébastien Mosser, Samira Ebrahimi-Kahou, Yann-Gaël Guéhéneuc	2022
P96	A multi-model based microservices identification approach	Mohamed Daoud, Asmae El Mezouari, Noura Faci, Djamal Benslimane, Zakaria Maamar, Aziz El Fazziki	2021
P97	Building a Performance Efficient Core Banking System Based on the Microservices Architecture	Fikri Aydemir, Fatih Başçiftçi	2022
P98	A DDD Approach Towards Automatic Migration To Microservices	Malak Saidi, Anis Tissaoui, Sami Faiz	2023
P99	Automated Planning for Software Architectural Migration	Nacha Chondamrongkul, Jing Sun, Ian Warren	2020
P100	Design principles, architectural smells and refactorings for microservices: A multivocal review	Davide Neri, Jacopo Soldani, Olaf Zimmermann, Antonio Brogi	2020
P101	Improving Industry 4.0 Readiness: Monolith Application Refactoring using Graph Attention Networks	Tanisha Rathod, Christina Terese Joseph, John Paul Martin	2023
P102	Incremental analysis of legacy applications using knowledge graphs for application modernization	Saravanan Krishnan, Alex Mathai, Amith Singhee, Atul Kumar, Shivali Agarwal, Keerthi Narayan Raghunath, David Wenk	2022
P103	Keep it in Sync! Consistency Approaches for Microservices – An Insurance Case Study	Arne Koschel, Andreas Hausotter, Moritz Lange, Sina Gottwald	2020
P104	Microservice remodularisation of monolithic enterprise systems for embedding in industrial IoT networks	Adambarage Anuruddha Chathuranga De Alwis, Alistair Barros, Colin Fidge, Artem Polyvyanyy	2021
P105	An Approach to Break Down a Monolithic App into Microservices	Taimoor Syed, Jun Long, Vijay Khatri, Mansoor Khuhro	2021

Table A1 continued

ID	Title	Author	Year
P106	A Novel Methodology to Restructure Legacy Application onto Micro-Service-Based Architecture System	T.R. Vinay, Ajeet A. Chikkamannur	2022
P107	Impacts of Decomposition Techniques on Performance and Latency of Microservices	Chaitanya K. Rudrabhatla	2020
P108	Remodularization analysis for microservice discovery using syntactic and semantic clustering	Adambarage Anuruddha Chathuranga De Alwis, Alistair Barros, Colin Fidge, Artem Polyvyanyy	2020
P109	Analysis and Development of Microservices Architecture in Loan Application System of Cooperative Enterprise in Indonesia	Reynaldi Lie, Ahmad Nurul Fajar	2022

Appendix B. Publication venues

Table B1. Publication venues of the primary research papers

Publication venue	#	ID
IEEE International Conference on Software Architecture Companion (ICSA-C)	4	P11, P34, P41, P54
IEEE Software	3	P1, P9, P10
IEEE International Conference on Software Architecture (ICSA)	3	P26, P61, P88
Euromicro Conference on Software Engineering and Advanced Applications (SEAA)	3	P38, P60, P66
International Journal of Advanced Computer Science and Applications (IJACSA)	3	P51, P63, P107
Software: Practice and Experience	2	P14, P64
On the Move to Meaningful Internet Systems: OTM Workshops	2	P16, P43
International Journal of Computer Applications (IJCA)	2	P24, P92
Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (DEVOPS)	2	P25, P29
Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS)	2	P42, P89
Empirical Software Engineering (EMSE)	2	P56, P59
International Conference on Advanced Information Systems Engineering (CAiSE)	2	P104, P108
Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems	1	P2
Conferencia Latinoamericana En Informatica (CLEI)	1	P3
Baltic DB&IS Conference Forum and Doctoral Consortium	1	P4
Advances in Service-Oriented and Cloud Computing (ESOCC)	1	P5
Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE)	1	P6
Open Conference of Electrical, Electronic and Information Sciences (eStream)	1	P7
Enterprise Modelling and Information Systems Architectures (EMISAJ)	1	P8
Computers in Railways XVII	1	P12
Asia-Pacific Software Engineering Conference (APSEC)	1	P13
Microservices	1	P15
Advances on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)	1	P17
International Systems and Software Product Line Conference (SPLC)	1	P18
European Conference on Software Architecture (ECSA)	1	P19

Table B1 continued

Publication venue	#	ID
Journal of Systems and Software (JSS)	1	P20
IEEE Transactions on Services Computing	1	P21
Software Engineering in the Era of Cloud Computing	1	P22
Service-Oriented Computing (ICSOC)	1	P23
International Conference on Information Technology-New Generations (ITNG)	1	P27
IEEE International Conference on Software Maintenance and Evolution (ICSME)	1	P28
Brazilian Symposium on Software Engineering (SBES)	1	P30
International Workshop on Refactoring (IWoR)	1	P31
On the Move to Meaningful Internet Systems (OTM)	1	P32
Asia-Pacific Symposium on Internetware	1	P33
International Conference of the Chilean Computer Science Society (SCCC)	1	P35
Workshop on Software Visualization, Evolution, and Maintenance (VEM)	1	P36
Applied Informatics (ICAI)	1	P37
Smart Applications and Data Analysis (SADASC)	1	P39
International Conference on Evaluation and Assessment in Software Engineering (EASE)	1	P40
Second International Conference on Information Systems and Software Technologies (ICI2ST)	1	P44
Service Oriented Computing and Applications (SOCA)	1	P45
IEEE International Conference on Software Reliability Engineering Workshops (ISSRE Wksp)	1	P46
International Conference on Computational Science and Its Applications (ICCSA)	1	P47
The International Conference on Advances in Emerging Trends and Technologies (ICAETT)	1	P48
IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)	1	P49
IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)	1	P50
International Conference on Smart Computing and Informatics	1	P52
Information and Software Technology	1	P53
International Conference on Dependable Systems and Their Applications (DSA)	1	P55
IEEE Annual Computers, Software, and Applications Conference (COMPSAC)	1	P57
Journal of Engineering Management and Competitiveness (JEMC)	1	P58
International Journal on Advances in Software (IARIA)	1	P62
International Computer Symposium (ICS)	1	P65
Anais Da Escola Regional De Engenharia De Software (ERES)	1	P67
International Conference on Algorithms, Computing and Systems (ICACS)	1	P68
Journal of Web Engineering (JWE)	1	P69
International Conference Automatics and Informatics (ICAI)	1	P70
Computer Applications in Engineering Education	1	P71
Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)	1	P72
International Conference on Software Maintenance (ICSM)	1	P73
International Journal of Innovations in Science and Technology (IJIST)	1	P74
Applied Sciences	1	P75
International Conference on Web Engineering (ICWE)	1	P76
International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)	1	P77
Expert Systems	1	P78
Handbook of Re-Engineering Software Intensive Systems into Software Product Lines	1	P79
Product-Focused Software Process Improvement (PROFES)	1	P80
Software and Systems Modeling (SoSyM)	1	P81
Digitalization of society, business and management in a pandemic: Interdisciplinary Information Management Talks (IDIMT)	1	P82

Table B1 continued

Publication venue	#	ID
ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)	1	P83
IEEE Access	1	P84
Communications of the ACM (CACM)	1	P85
International Conference on Software Technologies (ICSOFT)	1	P86
IEEE Software	1	P87
IEEE Transactions on Software Engineering (TSE)	1	P90
Journal of Automation, Mobile Robotics and Intelligent Systems (JAMRIS)	1	P91
Journal of Object Technology (JOT)	1	P93
IEEE/ACM International Conference on Automated Software Engineering (ASE)	1	P94
Journal of Software: Evolution and Process	1	P95
Journal of Systems Architecture (JSA)	1	P96
Journal of Grid Computing	1	P97
International Conference on Advanced Systems and Electric Technologies (IC_ASET)	1	P98
IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)	1	P99
Software-Intensive Cyber-Physical Systems (SICS)	1	P100
IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)	1	P101
Joint International Conference on Data Science and Management of Data (CODS-COMAD)	1	P102
International Conference on Advanced Service Computing	1	P103
Sylwan Journal	1	P105
Emerging Research in Computing, Information, Communication and Applications (ERCICA)	1	P106
Journal of Theoretical and Applied Information Technology (JATIT)	1	P109

e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240105, DOI: 10.37190/e-Inf240105

Measuring End-user Developers' Episodic Experience of a Low-code Development Platform

A Preliminary Study

Dongmei Gao*, Fabian Fagerholm*

*Department of Computer Science, Aalto University

dongmei.gao@aalto.fi, fabian.fagerholm@aalto.fi

Abstract

Background: As low-code development platforms (LCDPs) are becoming a trend, understanding how end-user developers think and feel as they work with such platforms is important. Particularly, assessing experiences during episodes of use can contribute to overall experience throughout long-term use.

Aim: This paper aims to understand end-user developers' episodic experience when they are building an application on a low-code platform and to provide guidance on how such experiences can be measured.

Method: We designed the Episodic Developer Experience Questionnaire for LCDPs based on prior literature and refined it through expert Delphi sessions. The instrument contains 10 individual experience items, capturing various aspects of episodic experience. We further validated it through remote online tests on an LCDP.

Results: The results showed significant differences in the relationships between items describing aspects of overall experience and items describing perceptions of tool quality and task difficulty. Programming expertise also affected end-user developers' episodic experience.

Conclusion: The study illustrates the design of questionnaire-based experience assessment in the context of development and identifies the importance of separating personal experience from assessment of tasks and tools since tool quality and task difficulty do not necessarily influence experience straightforwardly.

Keywords: developer experience, episodic experience, low-code development platforms, experience measurement, software engineering, human-computer interaction

1. Introduction

Developer experience (DX) refers to the subjective experiences arising from the involvement in software development [1, 2]. Understanding and improving DX can contribute to better support and increased well-being for software developers and may ultimately lead to higher levels of creativity and productivity [3, 4]. Developer motivation and happiness – factors of a positive experience – are also regarded to be correlated with the success of a project [5, 6].

Software developers have often been understood as having expert programming knowledge, while end-user developers [7] may have different kinds of domain knowledge but

Submitted: 15 Jun. 2023; Revised: 15 Dec. 2023; Accepted: 5 Feb. 2024; Available online: 1 Mar. 2024

are not specialised in software development. End-user developers therefore need software development tools that allow them to generate software programs or applications using domain concepts without needing to delve into the details of the implementation. Recently, Low-Code Development Platforms (LCDPs) have emerged in the industry to enable domain experts without programming knowledge to build applications [8, 9]. LCDPs offer a variety of ready-made components and features that allow making applications easily, usually by manipulating visual objects. Because they allow rapid expression of ideas, even developers experienced with textual programming languages may prefer to use LCDPs to save effort and time for certain kinds of tasks. Given the diverse levels of knowledge among developers, some LCDPs are designed to support both inexperienced and expert developers. They aim to enable developers of all levels to achieve their goals and to give them a positive developer experience.

LCDPs are still new and it is unclear how widespread their use will become. Some recent studies have discussed the conceptual understanding of LCDPs [10, 11], their characteristics [8], and challenges and opportunities associated with their use [12]. To understand how LCDPs could fit into the tool landscape of developers at different skill levels, it is important to understand how they affect DX.

DX can be observed on different time scales, varying from a developer's earliest awareness of a tool to use over a long period of time [2]. In this study, we focus on episodic experience to understand how developers feel after a single session of LCDP use where they focus on a bounded task, such as developing an individual application feature, in a given time. Their experience may change over time as they use an interactive product. They may be excited to use a new product for the first time, or attracted by its appearance, then become anxious by the complexity of the interaction during use, or satisfied because of the intuitiveness and efficiency of the functionality, and finally develop a complex experience about the product after a longer period of use.

Measuring and tracking episodic experience can yield insights into how the long-term experience has formed. It can also inform iterative design improvements to the platform. Our study design works by eliciting episodic experiences through naturally-bounded sessions of work that can be completed within a given period. Particularly, each session has a small set of goals and utilises a particular set of tool features, which we believe is an important unit of observation to inform the understanding of DX and assist in improving the design of LCDPs and other development tools.

To facilitate the comparison of experiences of LCDP use, we propose EDEQ-LCDP, a questionnaire instrument that captures important dimensions of developers' episodic experience. The intent is to provide an instrument that can be used rapidly after a development session to give an indication of the contents and quality of the developer's experience. The development of the questionnaire also illustrates how measuring DX is different from measuring UX in two senses: in terms of what experiential content or characteristics are relevant to the context of development rather than the context of use, and in terms of the object of assessment, which can vary between the experience itself and an assessment of a tool or product. We seek to disentangle the experience from assessments of the tool and the task and discuss the challenges of doing so.

The aim of the present study is therefore to provide an understanding of end-user developers' episodic experience while performing application-building tasks on an LCDP so that the development of measurement instruments for episodic developer experience can be better guided. We formulated the following research question:

RQ: How can end-user developers' episodic experience of low-code development platforms be measured?

The paper also contributes to the EDEQ-LCDP instrument as a basis for episodic developer experience measurement. To our knowledge, the instrument proposed in this paper is the first to focus specifically on the episodic experience of LCDPs. We take some steps towards examining the construct validity of the questionnaire as well as its content validity for practical and conceptual purposes, while properties of statistical generalisability are of secondary importance. While our study is performed using a particular LCDP, the instrument is not tied to any of its special features, and could, bearing limitations in mind, be used in tasks with other LCDPs as well.

2. Related work

DX has been defined as "a means for capturing how developers think and feel about their activities within their working environments" [1]. Fagerholm and Münch [1] proposed a conceptual framework for developer experience with three main dimensions – cognition (perception of development infrastructure), affect (emotional state about work) and conation (the value of one's own contribution). An extended discussion of the three dimensions was provided in a study of measuring DX toward the use of a Deep Learning (DL) platform [13]. Here, "cognition" is the rational basis provided by the DL platform, "affection" is explained as the emotional state, and "conation" is about the developer's tendency to use the DL platform voluntarily.

Transferring from the concept of user experience (UX), Fagerholm and Münch [1] conceptualised DX as experiences arising from the context of development, with impacts for individual developers and teams of developers, but also for organisational and product outcomes such as process adherence, productivity, and the quality of the end product. Thus, DX is a concern for developers themselves and for the organisations they work in and has implications for the customers and end-users of the software they produce.

Time seems to have an impact on the importance people attribute to different qualities of the experience with interactive products [14]. *Anticipated experience* refers to evaluations that occur before use, *immediate* or *momentary experience* is related to in-the-moment impressions, *episodic experience* emerges during a specific time-bounded duration, and *accumulated experience* (also known as cumulative experience) happens after long-term usage. Fagerholm's theoretical framework also categorises developer experience into immediate, episodic, and cumulative experience [2].

Episodic experience has drawn attention in various contexts: for example, research on learning to become a coach has shown that it can affect the way that individuals perceive what they know, and have the potential to influence their perception of future learning [15]. Marti and Iacono [16] recorded users' experiences during four weeks of using a product, showing that their episodic experience changed over time. Despite the crucial importance of usability in the product's initial acceptance, aspects of reliability, motivation, comparison with other products, change in behaviour and touch points – how the product communicates with the user, for example by notifications and alerts – are even more crucial for a user to resonate with a product and value it in the long term.

Greiler et al. [17] proposed an actionable conceptual framework to better understand and improve DX. They suggest putting the framework to work using an Ask-Plan-Act process: make problems visible by qualitative or quantitative means, determine the area to

be improved, and make continuous and small improvements. A survey instrument could be utilised in such an improvement cycle.

To facilitate the development of a DX instrument for the LCDP context, we should first understand more about the factors influencing DX, learn more about the characteristics of end-user developers as a particular group of LCDP users, and finally describe current research on LCDPs themselves. We discuss these topics in the following sub-sections.

2.1. Factors influencing developer experience

While some attempts have been made to construct survey instruments to assess developer experience (e.g., the DEXI scale [18]), there is currently a lack of instruments designed to measure DX in specific settings and for particular categories of experience. Previous studies have aimed to identify metrics or factors that impact developer experience. Kuusinen [19] conducted a survey considering a particular cross-platform IDE and found characteristics such as efficiency of use, flexibility, informativeness, intuitiveness, and reliability to be appreciated by developers. While developers in this study concentrated mainly on the pragmatic qualities of the tool, they also related experiences of hedonic aspects, such as affection for the tool, and expressions of feeling at home with it and that it has a good atmosphere.

The importance of developers' motivation for work has been widely emphasised in software engineering [20]. Intrinsic factors, such as autonomy and sense of achievement, have been found to be essential motivators of software developers [5]. Graziotin et al. [21] investigated factors associated with developer unhappiness, finding several factors to negatively influence happiness, such as being stuck in problem-solving, time pressure, bad code quality and coding practices, mundane or repetitive tasks, imposed limitations on development, and personal issues. Storey et al. [4] found that developers' job satisfaction can in turn impact their perceived productivity. To improve developer satisfaction, they identified several factors: how managers manage, and whether developers can effectively use their skills and believe their work has an impact. Linberg [22] found a large gap between developers' definition of job success and traditional definitions. Developers might remain satisfied with their jobs even if the project seems like a failure by others. Motivation is thus a complex dimension of experience that may be very personal and not only be driven by external outcomes.

Previous research has presented numerous factors to describe developer experience in different contexts. Bobkowska [23] built a model to explain the intuitiveness of software engineering techniques using UX concepts. It includes four perspectives: cognitive processes (users' familiarity with the technique), motivations (using the technique to solve a problem), actions (actions lead to good or poor results) and emotions (positive or negative attitudes). The model attempts to explain how episodic experiences with the techniques build up into cumulative experience and a perception of intuitiveness. Here, the episodic experiences are related to specific actions taken within the frame of a particular technique, and those experiences then accumulate and are generalised into cumulative experiences. The latter may then include the notion of intuitiveness regarding the technique. However, intuitiveness may also be the starting point: early experiences influence motivation to learn, they influence affective attitudes towards the technique, and they moderate the results of learning and actions, which then set the direction for the development of further experiences.

Despite the many existing studies on cognitive (cf. [24]) and affective (cf. [25]) aspects involved in software development, how to measure episodic DX, such as what dimensions

to measure to capture the salient features of developers' experiences as compared to users' experiences, remains an open question.

2.2. End-user developers

End-user development (EUD) has been defined as "the set of methods, techniques, tools, and socio-technical environments that allow end users to act as professionals in those ICT-related domains in which they are not professionals, by creating, modifying, extending and testing digital artefacts without requiring knowledge in traditional software engineering techniques" [26]. In the context of EUD, the term end-user developer is also widely used to describe people who have neither the skills nor the interest to customise a system in the way that software professionals do [27].

Cotterman and Kumar [28] defined an end user as "any organizational unit or person who has an interaction with the computer-based information system as a consumer or producer/consumer of information". They mapped out three dimensions across which the role of users can vary: operation, development, and control. Operation refers to tasks and actions necessary for operating a computer system, development refers to tasks related to the development of such a system, and control refers to the decision-making authority to acquire, deploy, and use resources necessary for operation and development. The role of a user may then involve more or less of the activities across these dimensions, and according to this taxonomy, end users share some traits with persons whose primary role is to develop a system.

Today, the boundary between development and non-development tasks in the roles of employees who are not primarily hired as software developers can be considered somewhat fluid because of the flexible ways in which many software products can be customised or controlled programmatically after they have been deployed to users. In the case of LCDPs, this fluidity goes even further. End-users of LCDPs become end-user developers when they begin to develop their software applications using LCDPs, shifting their role across the development dimension of Cotterman and Kumar's taxonomy.

Most importantly, we assume that end-user developers range from professional developers to people who are not experienced with or specialised in conventional software development (so-called citizen developers) [29]. As far as we know, the term citizen developer first appeared alongside LCDP in a 2014 Forrester report [30]. Since then, the term has been widely embraced by LCDP vendors and organizations as their potential user groups [12]. The promise of LCDPs is that they allow citizen developers to build their software applications without the help of professional developers [11, 29] while professional developers can also benefit from LCDPs, for example by being more productive and reducing time spent on repetitive tasks. The low-code platforms on the market today are also increasingly focused on freeing people from heavy, monotonous underlying development tasks and focusing more on the realisation of ideas.

The two terms EUD and citizen developer thus have a significant overlap, with the latter originating in industry and focusing attention on the role of an employee in relation to the IT function of an organisation [29], and the former being a more long-standing and established term in research which we perceive as subsuming the other term. We have chosen to use the term EUD in this paper because of its longer history, broader and more elaborate definition, and because we do not investigate the aspects of citizen development that relate the citizen developer and the applications they produce to the larger IT infrastructure and governance in an organisation. We consider the end-user

developer as a person who has two roles, being an end-user of the product and a developer of an application at the same time, a definition that shares many important aspects of the citizen developer term and is consistent with terminology used in human-computer interaction and software engineering research (see, e.g., [26–29]).

2.3. Low-code development platforms (LCDPs)

In recent years, Low-Code Development Platforms (LCDPs) have emerged as a specific kind of EUP tool that reduces the effort of implementing simple applications. LCDPs can provide developers with the means to generate and deliver business applications quickly with minimum effort in terms of installation and configuration of the development environment, training, and implementation of the project [31]. They contain a set of components and features for programmers and non-programmers. LCDPs are primarily aimed at developers with a limited programming background [11]. These end-user developers rely on LCDPs to achieve their creative ideas without going through conventional development cycles to build applications from scratch as traditional developers do. In addition, faced with the ever-changing customer needs in the software business, firms often adopt LCDPs for fast delivery with a minimum of hand-coding, quick setup and deployment. They also address the ability of LCDPs to test business ideas within days or weeks [30]. Thus, professional developers (professionals with an education or career in software development) can benefit from LCDPs as well, for example by being more productive [29].

Waszkowski [31] pointed out three main features of LCDPs: databases, business processes and user interface. They simplify the development process, reduce the learning costs and visualise the coding environment to allow developers to spend less time on coding and focus on their goals. Although there is a varied range of different types of LCDPs, some typical functionalities and features can be identified [11]. Two major structures of LCDPs have been described: UI to Data and Data to UI [11], denoting the flow of design from either the user interface or the data model. Sahay et al. [11] list four LCDP characteristics that have a large impact on DX: 1) *interoperability*, by creating standards and a friendly ecosystem in a domain to mitigate issues caused by closed sources; 2) *extensibility*, allowing developers to customise capabilities to reduce the design constraints; 3) the *learning curve*, which is still less intuitive for end-user developers; and 4) *scalability*, such as running on the cloud to manage intensive computing [11].

The term no-code is closely related to low-code. It is sometimes used as a slight variation of low-code [10, 29] or a synonym to refer to low-code development practices [8]. As we can see in the market, LCDP vendors define their products as no-code or low-code tools according to their business goals. The distinction of them is still ambiguous and practitioners also use these two terms interchangeably. In this paper, we view the no-code platform as a subset of a low-code platform. End-user developers can use the basic functionality of the low-code platform without writing any code but are still required to write several lines of code to get access to more advanced features.

3. Research approach

In this paper, we present the development of a new questionnaire instrument that aims to identify important influencing factors of developers' episodic experience of LCDPs. The instrument we designed aims at end-user developers including both professional developers

and citizen developers. The purpose of developing the instrument is primarily to structure the phenomenon of developer experience in the context of LCDPs, to provide further understanding of it, and to provide guidance on how such experiences can be measured. To a lesser extent, we are also interested in the questionnaire instrument itself for purposes of measuring DX in an LCDP context, but we do not aim to fully validate the instrument for general use.

The process of developing the questionnaire and providing a preliminary validation of it involved three stages, as shown in Figure 1. First, we analysed related literature to collect potentially relevant question items and constructed a preliminary questionnaire based on these items (Stage 1). Second, we used a Delphi study [32] to gather opinions from a panel of experts, helping us to capture important nuances and refine the questionnaire (Stage 2). In the third stage, we ran a task-based test in which 19 developers were asked to use an LCDP in individual sessions (Stage 3). After each task, they filled in our questionnaire, providing us with data to further analyse it. The result of this research process is the final questionnaire proposed in this paper.

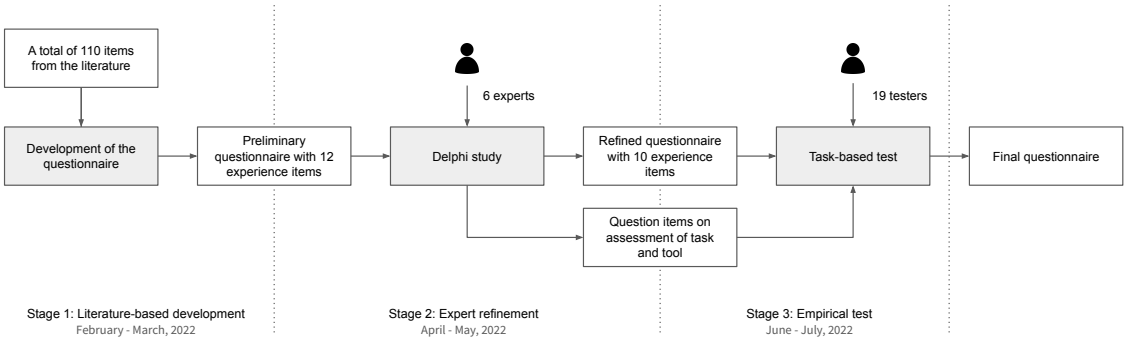


Figure 1. Overview of the research approach

The questionnaire aims to capture end-user developers' episodic experiences in relation to tasks performed using an LCDP. Assessing episodes with a limited number of tasks that involve known features enables assessment of specific parts of an LCDP, which may be helpful when using the questionnaire for design improvements in the tool.

We assume that the degree of familiarity with software development could influence individual developers' experience. Their prior familiarity can influence how much effort and time they need to invest in learning the platform. Therefore, the impact of prior programming background was taken into account in this study by introducing a set of background questions for the participants of the task-based test. Another factor that could influence a developer's experience is the complexity of the task and the environment in which it needs to be conducted. For example, a complex IT environment with many existing applications made using an LCDP presents a more challenging environment because the developer may have to consider existing software when they develop something new. However, in this study, we focus on the individual experience of an LCDP in a bounded development task. The questionnaire instrument does not assess the difficulty of the environment but aims to capture basic information about a developer's personal experience. To differentiate between the experience of the platform and the task, we included question items that asked for assessments of the tool and the task. We discuss the details and output of each stage in the following sections.

4. Literature-based development of a preliminary questionnaire

We developed an initial questionnaire by collecting experience items from existing studies with relevance to DX. Our starting point was a set of factors reported in a multi-vocal literature review of LCDP, which collected factors both from scientific and grey literature [33]. We also searched for articles which presented questionnaire instruments to measure DX or UX using Google Scholar and used items from those questionnaires as potential candidate items for our questionnaire.

4.1. Source literature

Nylund [33] presented factors that improve and worsen DX based on existing literature. The list of 17 DX influencers includes topics such as mood and feelings, project onboarding, and factors related to the software development methodology, such as Agile software development. This study facilitated our search for literature associated with DX by providing a systematically collected set of relevant literature. We expanded this set by a search for DX and UX measurement instruments. We first used Google Scholar with the keywords "developer experience measurement" to find additional candidate items. We also used the keywords "user experience measurement" to extend the search scope. We selected papers that presented questionnaire-based instruments that could potentially be relevant for measuring different aspects of developer experience.

We included items from the Developer Experience Scale (DEXI) [18], one of the first questionnaire instruments that aim to measure the DX of development tools. Its word pairs were obtained from several sources and selected based on their relevance to software development. It contains one item for measuring general quality, five items for pragmatic quality, and six items for hedonic quality. We also collected 20 items from a recent study conducted by Lee and Pan [13] which aimed to evaluate sub-constructs of DX – cognitive, emotional and behavioural – and which itself is based on items from various sources.

We found only a few instruments intended specifically for DX measurement. Because of the commonalities between UX and DX and the greater breadth and depth of UX research, we also considered other widely used UX surveys: the Short Dispositional Flow State Scale (SDFS-2) [34], Intrinsic Motivation Inventory (IMI) [35], and Short AttrakDiff-2 (SAD-2) [36].

4.2. Item selection and categorisation

We initially considered all items from the candidate questionnaires and proceeded to exclude items that did not fit our purpose. We de-duplicated items, removing those that were the same in two or more of the original questionnaires. We retained items that concerned episodic experience and discarded items that were referred only to immediate (e.g., "The appearance is beautiful.") and cumulative experience (e.g., "I will recommend it to my friends.").

Under the episodic experience category, we continued to subdivide the items by cognition, affect and conation to ensure that the instrument would cover the three aspects of the DX framework [1]. To perform this subdivision, we used the following guidelines:

1. *Cognition:* Items related to performing tasks and the efficiency or effectiveness of performing them, or to the usefulness of a tool in helping to perform them.
2. *Affect:* Items that describe emotions in a situation or related to a task or activity.
3. *Conation:* Items that describe the result or value obtained by a developer or that are related to developers' motivation or volition.

We did not assume that the final questionnaire should consist of items that are possible to map exactly to a single dimension in the framework. The purpose of this stage was to ensure a good starting point for the following steps in the questionnaire development.

At this stage, we wanted respondents' responses to focus only on their own experiences with the application development process, avoiding comments and perceptions about the tool or the task. Personal experience is complicated and can be influenced by various factors. Considering the aims of the questionnaire, it is more important to capture developers' notions of the overall experience of making an application than perceptions of the tool itself or the task they have just completed. For example, they might be satisfied with the user interface of the tool but anxious about the slow speed at which they progress in making their application; or they might feel that the platform is hard to use but the task in itself is easy to complete. With our focus being on individual developers' experience, we wanted the questionnaire to capture their assessment of their internal experience and not their observations of the tool or task. Also, focusing on the experience of the process of developing an application makes the questionnaire more general and not tied to any specific features of an LCDP. For these reasons, we removed items that referred to features or characteristics of tools, such as "It has explicit guidelines." and "It has a high level of information."

4.3. Questionnaire format

Since our questionnaire is meant to be used after episodes that may range from minutes to a few hours at most, we wanted a format that is quick to fill in. To that end, we adopted a semantic differential format (polar word pairs), following some existing and widely used UX questionnaires such as AttrakDiff-2 [36]. We also wanted a granular enough scale to let respondents capture nuances between the polar ends of each item. Psychometric literature suggests that more scale points provide better statistical properties for the data [37], but this has to be balanced with complexity for participants. While the issue can be debated, we opted for a seven-point scale based on current psychometric research (see, e.g., Taherdoost [38] for a summary).

The outcome of the preliminary design stage was a set of 12 items. These are available in the supplementary material [39].

5. Delphi study and final questionnaire

Delphi studies aim to improve decision-making by seeking the most reliable consensus from a group of experts [32]. The preliminary questionnaire described above was based on a literature review and our subjective analysis. We sought strengthened confidence in content validity, i.e., that our questionnaire items were consistent with the nature of LCDPs, and whether the wording was understandable in a practical setting. To this end, we contacted prospective experts through our industry collaboration networks and a purposive search on LinkedIn. Lynn [40] suggested using at least three and at most ten panellists for evaluating content validity. We selected and invited six experts with technical and research backgrounds to comment on the questionnaire. All participants were interested in DX and had work responsibilities related to the topic. Three experts work as UX Designers for an LCDP company; one is a principal user researcher in global large-scale research programs;

and the remaining two have rich experience with managing developers in industry practice. The participants are based in the Czech Republic, Finland, France, and the USA.

We held online meetings where we presented the in-progress questionnaire items to each expert in turn. They were free to make any comments and suggestions on the items, and we asked them open-ended questions to gain knowledge of how they perceived the items and why. After each session, we reviewed the opinions collected and refined the questionnaire. The new questionnaire was presented in the next Delphi session. In the last two iterations, the experts had no new remarks about the questionnaire that would have warranted further changes, i.e., we reached a point of saturation.

In the final questionnaire, we added two additional sections to specifically capture perceptions of the tool and the task separately from the respondent's overall individual experience.

The final questionnaire is shown in Table 1. The questionnaire's main instruction is to choose the most appropriate description for one's experience. The word pair items in the first part each capture an aspect of the respondent's experience. The questionnaire does not attempt to assess, for example, the details of the LCDP or the product being developed using it. Parts 2 and 3 are included to provide a way to separately assess aspects of the tool and task. They do not have to be used if this information is not needed. In this paper, they are used for the preliminary validation of the questionnaire.

Table 1. Final questionnaire used in the experiment of measuring end-user developers' episodic experience of LCDPs. Parts 2 and 3 are control items, while part 1 constitutes the developer experience instrument

Part 1: Episodic Developer Experience Questionnaire for Low-code Development Platforms (EDEQ-LCDP).			
Think about the session you just completed. Please enter what you consider the most appropriate description **for your experience** during the session.			
1.	Smooth	○ ○ ○ ○ ○ ○ ○	Exhausting
2.	Easy	○ ○ ○ ○ ○ ○ ○	Difficult
3.	Frustrating	○ ○ ○ ○ ○ ○ ○	Enjoyable
4.	Satisfying	○ ○ ○ ○ ○ ○ ○	Dissatisfying
5.	Distracted	○ ○ ○ ○ ○ ○ ○	Concentrated
6.	In control	○ ○ ○ ○ ○ ○ ○	Lack of control
7.	Exciting	○ ○ ○ ○ ○ ○ ○	Boring
8.	Slow	○ ○ ○ ○ ○ ○ ○	Quick
9.	Free to explore	○ ○ ○ ○ ○ ○ ○	Limited
10.	Productive	○ ○ ○ ○ ○ ○ ○	Unproductive
Part 2: How much do you agree with the following statements regarding **the tool**.			
		Completely Disagree – Completely Agree	
1.	The tool was reliable	○ ○ ○ ○ ○ ○ ○	
2.	The user interface was consistent	○ ○ ○ ○ ○ ○ ○	
3.	The tool was quick to respond	○ ○ ○ ○ ○ ○ ○	
Part 3: How much do you agree with the following statements regarding **the task**.			
		Completely Disagree – Completely Agree	
1.	The task was easy to understand	○ ○ ○ ○ ○ ○ ○	
2.	The task was easy to complete	○ ○ ○ ○ ○ ○ ○	

6. Task-based test

To gain insights into how our instrument works when applied to practical settings, we performed a task-based experiment with 19 participants. To identify the complexity and the acceptability of the tasks, we first conducted a pilot study with three participants in which we tested and refined the tasks iteratively. We analysed the resulting data using statistical methods to gain an initial picture of its suitability for field use.

6.1. Participants

One of the ambitions of LCDPs is to enable end-user developers to build their own applications without much effort. The learning cost and effort for professional and non-professional developers are generally not the same. We therefore created a screening survey to obtain two groups of participants, one of which had 14 participants with a programming background and the other consisting of 5 participants without. The entire set of participants included 10 students and 9 company employees. They were all fluent in English and mainly from Asia, Europe, Africa and the USA. Their fields of work or study included: research (6), software development (3), engineering (4), design (3), teaching (1), materials (1), and new media (1). The participants participated on the condition of anonymity and were compensated with a gift card worth 20 €.

6.2. Procedure

The test sessions were performed online, using SAP AppGyver[1], which is marketed as a no-code development platform for building mobile apps for both Android and iOS. As mentioned before, we consider no-code platforms to be a subset of low-code platforms. The LCDP's positioning of itself in the marketplace does not make a clear distinction between low- and no-code. For example, in AppGyver's case, some formula functions require a basic knowledge of programming to read and write, such as data variables and IF-statements. AppGyver provides clear documentation, but we found in our testing that some participants still found it difficult.

Participants shared their screens to allow us to observe their use of the tool. Each session lasted one hour and consisted of three parts:

- **Introduction and guided tour (20 minutes):** We explained the study procedure, what data would be collected and how it would be used and retained, and obtained informed consent following our university research ethics guidelines.
- **Warm-up task (5 minutes):** Since most of the participants had never used LCDPs, they might feel confused or nervous about using a new tool for the first time. We therefore used a warm-up task to help them get familiar with the tool. In this task, participants were instructed to create the user interface of a simple to-do list application that displayed a list of tasks, an input field and a button with the text "Add task".
- **Two independent tasks (35 minutes):** Then, in the main study tasks, participants added simple functions to the user interface of the to-do application created earlier to make it interactive, including the functionalities "Delete task from to-do list" (study task 1) and "Add new task to to-do list" (study task 2). After each of these two tasks, participants filled in our DX questionnaire using an online form.

[1] https://www.appgyver.com/

6.3. Analysis

We first identified differences in how participants felt across tasks and then used the Mann–Whitney U Test and Kendall's Tau to analyse the correlations between individual experience items and various perspectives of tasks and the tool. After that, we used the Intraclass Correlation Coefficient (ICC) to test the consistency among participants in the same group.

6.3.1. Task difficulty vs. developer experience

By comparing the scoring details of the two tasks, we made a simple assessment of the validity of the scoring. Concerning task completion, 12/19 people completed task 1, with 5 of them following the prompts. 5/19 succeeded in task 2, but only 2 of them completed it independently. Table 2 presents the average scores of tool quality and task easiness. Responses to *The task was easy to complete* indicate that task 1 was easier to complete than task 2. This finding is consistent with our intended task design. In contrast, participants considered task 2 to be easier to understand. This can be explained by the learning effect: when doing the first task, participants needed time to understand the task, but due to the continuity and similarity of the tasks, they could be more confident about the second task. In terms of individual experience factors, all experience factors got higher scores in task 1 except for *Satisfying – Dissatisfying*. Participants might be more satisfied about their performance during the process if they solved a complex problem.

Table 2. Average score of tool quality and task easiness in tasks 1 and 2. $N = 19$

Item	Task 1	Task 2
The task was easy to understand.	4.842	5
The task was easy to complete.	4.158	3.211
The tool was reliable.	5.053	5.105
The user interface was consistent.	5.158	4.842
The tool was quick to respond.	5	5.368

6.3.2. Individual experience items vs. the evaluation of the tool and tasks

We ran a Mann–Whitney U Test to compare whether participants who evaluated the platform or the task highly and those who evaluated them low had significantly different experiences. We compared the responses of those developers who assessed these factors on a seven-point scale of 5 or higher and those who assessed them as 3 or lower. Notably, we tested tasks 1 and 2 separately to avoid bias since they were different. However, for the platform, there were no statistically significant differences between those evaluation perspectives (tool reliability, user interface consistency and tool response) in either task. Even if there may be significant differences in one of the tasks, they do not apply to the other. For example, in task 2, those respondents with high experience satisfaction perceived the platform to be highly reliable more often than those with low experience satisfaction, whereas this was not evident in task 1. This indicated the difficulty of the tasks played an essential role in measuring episodic experience.

The same applies to the evaluation of the understanding of the task. None of the experience items were able to distinguish between users who found it easy or difficult to

Table 3. Statistically significant Mann–Whitney U test results between respondent groups Task Completion$_{good}$ and Task Completion$_{bad}$ for all items. n.s. = not significant. $N = 19$

	Task 1		Task 2	
Item	U	p	U	p
Easy – Difficult	8	<0.01	9.5	<0.05
Free to explore – Limited	15.5	<0.05	14	n.s.
In control – Lack of control	15.5	<0.05	8	<0.05
Exciting – Boring	10.5	<0.05	10.5	<0.05
Satisfying – Dissatisfying	32.5	n.s.	5.5	<0.01
Quick – Slow	16.5	<0.05	16	n.s.
Concentrated – Distracted	22.5	n.s.	5.5	<0.01
Enjoyable – Frustrating	22.5	n.s.	0.5	<0.001

understand either task, although four experience items could differentiate users in task 1. As for participants who found the task easy to complete and those who found it difficult, 3 out of 10 word pairs (*easy*, *in control* or *excited*; see Table 3) supported the separation between them. This indicates that when participants rated these items high after performing a task using the LCDP, they were also more likely to find the task easy to complete.

We used Kendall's Tau Correlation analysis to discover which items in each scale were significantly correlated with each perspective.

From the results, few items had strong correlations with user interface consistency and tool response. Since the LCDP used in this test had already been subjected to user testing and was already a mature product, it was not surprising that these two items were not strongly correlated with differences in individual experience; they may have captured more objective traits of the tool. However, two other items, tool reliability and task completion, reflected strong correlations with a majority of individual experience items:

Tool reliability. Correlations between tool reliability and individual experience items are presented in Table 4. 7/10 (70%) of word pairs correlated with tool reliability in task 1 while 6/10 (60%) had significant correlations with tool reliability in task 2. What is notable is that 4/10 (40%) of all experience items had significant correlations with tool reliability both in tasks 1 and 2. All experience items could predict tool reliability either in tasks 1 or 2 except for *Smooth – Exhausting* which only showed a strong correlation with the ease of task completion in task 2. In addition to the possibility that it had little to do with the platform and the task, we also speculate that it was the result of a more diverse user understanding of this experience factor.

Table 4. Results of Kendall's Tau correlation analysis between the tool was reliable assessment and individual developer experience items. n.s. = not significant. $N = 19$

	Task 1		Task 2	
Item	τ	p	τ	p
Easy – Difficult	0.434	<0.05	0.301	n.s.
Free to explore – Limited	0.331	n.s.	0.369	<0.05
Satisfying – Dissatisfying	0.331	n.s.	0.369	<0.05
Productive – Unproductive	0.507	<0.01	0.227	n.s.
In control – Lack of control	0.49	<0.01	0.479	<0.01
Exciting – Boring	0.545	<0.01	0.505	<0.01
Enjoyable – Frustrating	0.514	<0.01	0.597	<0.01
Concentrated – Distracted	0.407	<0.05	0.509	<0.01
Quick – Slow	0.418	<0.05	0.318	n.s.

Table 5. Results of Kendall's Tau correlation analysis between the task was easy to complete and individual developer experience items. n.s. = not significant. $N = 19$

	Task 1		Task 2	
Item	τ	p	τ	p
Smooth – Exhausting	0.298	n.s.	0.399	<0.05
Easy – Difficult	0.583	<0.01	0.363	n.s.
Free to explore – Limited	0.389	<0.05	0.37	<0.05
Satisfying – Dissatisfying	0.312	n.s.	0.644	<0.01
Productive – Unproductive	0.465	<0.05	0.372	<0.05
In control – Lack of control	0.495	<0.01	0.372	<0.05
Exciting – Boring	0.576	<0.01	0.519	<0.01
Enjoyable – Frustrating	0.389	<0.05	0.725	<0.01
Concentrated – Distracted	0.265	n.s.	0.412	<0.05
Quick – Slow	0.475	<0.05	0.478	<0.05

Task completion. 7/10 (70%) and 9/10 (90%) of items significantly correlated with task completion in tasks 1 and 2. All items had significant correlations with task completion in both tasks 1 and 2 (Table 5). It is worth noting that compared to other items, *Satisfying – Dissatisfying* and *Enjoyable – Frustrating* had a stronger relationship ($\tau = 0.644$ and $\tau = 0.725$, respectively) with task completion in task 2. The more participants felt confident completing the task, the more satisfied and enjoyable they felt especially when the task was hard.

With the Mann–Whitney U test and Kendall's Tau correlation analysis, we realized some experience items were changeable across the two tasks. The difficulty of tasks seems to play a great role in measuring developer experience. In further research or validation, a comparison of more tasks of different difficulties together would be advisable.

6.3.3. Programming backgrounds vs. developer experience

Intraclass Correlation Coefficient (ICC) was used to find out if there was consistency in the results of participants with a similar programming background. Table 6 shows the level of consistency in the performance of different groups. According to statistical guidelines [41, 42], ICC values higher than 0.50 are acceptable. Therefore, the experience of participants without programming experience showed consistency in both tasks. Conversely, the participants who had programming experience exhibited inconsistent performance across the two tasks.

Thus, programming experience does make a difference to the episodic experience of the LCDP, but this effect is varied. Our participants' programming expertise, such as programming language and skills, was diverse, indicating that the quantitative findings

Table 6. Results of Intraclass Correlation Coefficient for each two groups measuring the experience. n.s. = not significant

	Task 1		Task 2	
Group	ICC	p	ICC	p
A: No programming experience	0.582	<0.001	0.626	<0.001
B: <3 years of programming experience	0.093	<0.001	0.185	<0.001
C: ≥3 years of programming experience	0.253	n.s.	−0.160	n.s.
B and C	0.076	<0.01	0.150	<0.001

make intuitive sense. For example, one participant was mainly responsible for data analysis using existing libraries while another focused on algorithmic programming. Developers writing operational logic and visual user interfaces also had diverse understandings of workflow. Most of them thought their prior programming expertise did not contribute to their usage of the LCDP. But one participant – a front-end developer – found it easy to do the task because the logic was almost the same. More experiments are needed to discover the relationship between different programming expertise and the experience of using an LCDP.

7. Discussion

In the previous sections, we have described how we created a preliminary questionnaire instrument for measuring episodic developer experience during LCDP use based on analysis of prior literature, how we refined this questionnaire using a Delphi study with experts to obtain a final questionnaire (EDEQ-LCDP), and how this questionnaire was used in a task-based test.

Our research question asks how to measure the end-user developers' episodic experience of a low-code development platform. We designed a questionnaire with 10 items (Part 1 in Table 1) to answer it. Each question item measures a specific aspect of the episodic experience. The items in parts 2 and 3 are used to provide a way to separate personal experience from the assessment of the LCDP and the tasks, and they do not have to be used in practice. In the process of developing the questionnaire instrument, we suggested separating experience from task difficulty and tool usability because experience is quite complicated and affected by various aspects, such as tool, task or even the working environments. The experience may consist of positive and negative assessments independent of tool usability and task difficulty. We tried to free respondents from identifying their precise feelings. Furthermore, we compared our work with related work and discussed the differences between episodic experience and cumulative experience.

7.1. Experience should be set apart from task difficulty and tool usability

The questionnaire we designed covers 10 items to measure developer experience after an episode of completing tasks in an LCDP. All of them are related to developers' thoughts and feelings. We tried to separate the personal experience of an application development process from the task and the tool. During the study, some experts and participants asked us what an item referred to, such as whether *Satisfying – Dissatisfying* meant that the tool was satisfying or that developers felt satisfied with their performance during the process.

We aimed to focus on participants' overall feelings when they were performing the task. Some people might find a tool novel and helpful, but feel dissatisfied with their performance, perhaps because they were unfamiliar with this type of platform or were doubting their ability to perform the task. As a result, their experience of the process was unsatisfactory. We were more interested in their overall personal experience than asking them directly about their attitude to the platform or the task.

Each personal experience factor might be affected by multiple factors. For example, in this study, *Easy – Difficult* did not show a significant correlation with *The task was easy to complete*. One participant gave task difficulty the lowest score but scored their experience highly difficult. This participant explained that although they personally felt it was difficult

to perform the task, they still considered the task to be simple in general and that their lack of expertise was the main reason for feeling it was difficult. If they would have a second chance to do the task, they thought they would perform better. Another participant also found the task easy, but felt nervous about being asked to complete it within a specific time. They stated that given free time, they would look up the documentation carefully.

These examples illustrate the importance of capturing the overall personal experience of development episodes, and keeping this separate from usability assessments of tools. Tool providers usually aim to create a better experience for tool users – in this case, developers using an LCDP. It is not enough to obtain assessments of the product's appearance, functionality, help manuals, etc.; only if developers obtain positive experiences, which may depend on a multitude of personal reasons, and if they feel happy, satisfied, and have achieved their goals using the product, they can become inclined to continue using it and recommend it to their peers.

7.2. Comparing to related work

Our DX instrument was based on prior literature and it bears resemblance to many UX measurement instruments. However, most UX instruments that we are aware of focus on the respondent's assessment *of the product* rather than *of their personal experience*. There are also some existing instruments specifically for measuring DX, but they are not specifically designed for episodic experience. We discuss these issues using a few examples of existing questionnaires below.

The User Experience Questionnaire (UEQ) [43] contains 26 items including six dimensions: Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation, and Novelty. It aims to be quick and easy to apply and is suggested to complement the use of other evaluation methods with subjective quality ratings. In contrast to our questionnaire, it is focused more on an assessment of the *product* than the experience. For instance, the UEQ item *Usual – Leading edge* describes the technology used in the product. *Conventional – Inventive* stresses the novelty of the product. The items thus represent the respondent's personal assessment of the product but do not necessarily reflect an assessment of their personal experience with using the product in a concrete task episode.

Similarly, AttrakDiff-2 aims to capture product perceptions and evaluations [36]. It consists of items to capture pragmatic and hedonic quality as well as general quality. With items such as *confusing – structured* (pragmatic), *dull – captivating* (hedonic), and *good – bad* (general), it appears to be asking for assessments of different aspects of product quality. The respondent's experiences with the product can be assumed to affect these assessments, but as with the UEQ, what is asked for is an opinion regarding the product, not an account of the respondent's personal experience with using it.

Many other UX measurement instruments follow the same approach. Our instrument takes steps to focus the respondent's attention on their personal experience by instructing them to think about the development episode they just performed and to provide "a description for your experience". While they may factor in aspects of the usability of the tool or the difficulty of the task, this is by intention: the experience of the previous development episode is based on a multitude of factors but what we wish to gain insight into are the contents of the participant's near-term episodic memory of that episode.

Another aspect of the present study is that it focuses on the context of development, where participants are creating new software using a development tool. While DX can be considered a sub-field of UX, the experience of developers has its own specifics that should be

taken into account and that are relevant also to inform the management and organisation of software development in organisations. For example, developers often have particular wishes when it comes to tool choice, such as the extent to which they are customisable according to the developer's wishes, often with scripts or plug-ins that can extend or alter its functionality. They may also have preferences regarding the way a tool allows them to express their ideas, and they may want to know what "goes on under the hood", i.e., they may be concerned about their ability to control the final product when using a high-level tool that hides implementation details. Such concerns are less typical for end-users who are not producing new software but rather work with data such as text or images where the result is what they see on their screen. For these reasons, we have taken the characteristics of development tools and their use in both an organisational and individual setting in our questionnaire.

To our knowledge, instruments for specifically measuring DX are rare. However, some do exist. The DEXI scale is meant specifically for software development as a general questionnaire for measuring DX of a GUI designer tool [18]. DEXI was formed based on frequently used UX questionnaires by selecting items from them that are appropriate for describing aspects of DX. Its *Limited – Extensive* is an example of capturing the customizability aspect of the developer's tools to give them as much freedom as possible. We used *Limited – Free to explore* to characterize this kind of experience; the wording was informed by our expert feedback. DEXI is similar to the UX questionnaires discussed above in that it focuses on the participant's assessment of the product – in the case of DEXI, an IDE. It thus does not address the participant's personal experience directly, but rather incorporates it into the respondent's assessment of the tool. In our view, this has at least practical implications for the use of the questionnaires to inform DX improvements. Whereas DEXI directs the focus towards the tool itself, our questionnaire invites a discussion about the participant's experience in general, which may reveal issues regarding the tool, but could also lead to discovering issues related to the task, the participant's perception of themselves as a developer, or relationships between individual, task, and tool. While it is not guaranteed that a participant will discuss issues beyond the tool, our instrument does avoid directing attention primarily towards the tool. Depending on the purpose, this may or may not be beneficial.

In other work, Lee and Pan [13] constructed a set of items to measure DX of deep-learning platforms based on both the conceptual model of DX [1] and related research on customer experience in fields such as marketing. Their questionnaire uses a different format with statements such as "The interaction between the platform and the developer does not require much mental effort" (cognitive), "The platform is attractive" (affective) and "Developers intend to use the platform" (conative). Respondents indicate agreement on a seven-point scale. Beyond the obvious difference of using a statement format, this questionnaire is much more general than ours and tries to map each item specifically to one dimension of the DX framework. It also appears to mix different levels of experience, with items regarding mainly cumulative and anticipated experience; the conative item here is an example of the latter. Our instrument is more focused specifically on LCDPs.

7.3. Episodic experience vs. cumulative experience

As noted above, previous DX questionnaires are not specific to episodic experience. Instead, the majority of research into DX instruments has focused to a large extent on cumulative experience, with some introduction of anticipated experience.

Studying cumulative experience makes sense when there is reason to believe that enough time has passed for such experiences to form, such as when people have used a product or tool for a long time. The main points of concern with regard to cumulative experience are different from the concerns of the first few times people use a product to solve a problem or accomplish a task. Taking the view that cumulative experience is a function of experiences over several episodes, one can assume that there is an overlap between cumulative and episodic experiences. However, previous research constructing DX instruments hardly distinguishes the two specifically.

Factors such as cost, backward and forward compatibility, version migration paths, and the surrounding developer community are important for the cumulative experience related to development tools, while such factors play a smaller role in the episodic experience. Motivation is another rather different aspect between the two levels of experience. During a development episode, developers seek a solution to a small set of specific problems while they may hope to achieve a bigger goal during long-term usage of the platform, including goals that span beyond an individual piece of software they are developing. For example, people's intention to use an LCDP is to build an application, which is a long-term goal. The goal will be achieved by several episodic goals which are to implement various features of the application. Beyond these, they may have long-term motivations such as advancing in their career. We thus argue that DX questionnaires should be designed with the different levels of experience in mind: general DX questionnaires focusing on cumulative experience have to consider factors beyond the tool, while instruments focusing on episodic experience can take a more narrow set of factors into account.

We believe that motivation also influenced the result of the test in our study. Participants did not have expectations towards this platform since it was their first time using it. They were unlikely to have an emotional attachment to the platform at the outset and were just required to perform the task whether or not it was inherently interesting to them. We therefore did not include general factors such as *Cheap – Premium* (from AttrakDiff-2) or *"Even if the platform price is high compared to other platforms, I think that developers should use the current platform"* (from Lee and Pan). We believe that such general and circumstantial factors are not of primary relevance to episodic experience, but they may work as part of assessments of anticipated or cumulative experience. However, developers, even end-user developers using LCDPs, may not be involved in tool purchase decisions so the cost of acquiring such tools may not be relevant to them.

Concentrated – Distracted is the only item that was strongly correlated with nearly all aspects of the platform in both tasks 1 and 2. Users' attention during an event influences the time, efficiency and completion of the task, therefore influencing their feelings. Concentration on task has been associated with flow experiences in prior literature (e.g., [34]), but it was not significantly correlated with developers' overall experience in assessing DX of a GUI designer tool [18]. Thus, the role of attention in episodic and cumulative developer experiences should be further investigated.

7.4. Insights for further research

This study provides a starting point for further research. Our statistical analysis of the questionnaire responses in our study gives preliminary indications that should be investigated further. The results of the Mann–Whitney U Test indicated that *Easy – Difficult*, *In control – Lack of control* and *Exciting – Boring* can differentiate between participant groups with low and high Task Completion Difficulty assessment. Of these three items, the

latter two were also significantly correlated with the ease of task completion (Kendall's Tau Correlation Analysis). In addition, the correlation analysis revealed an additional four items that had similar correlation: *Free to explore – Limited*, *Productive – Unproductive*, *Concentrated – Distracted* and *Quick – Slow*. Task Understanding was strongly correlated with *Productive – Unproductive* and *Quick – Slow*.

A number of items were correlated with the reliability of the platform although we could not successfully differentiate between participant groups based on these items. For example, *In control – Lack of control*, *Exciting – Boring*, *Enjoyable – Frustrating* and *Concentrated – Distracted* are associated with tool reliability. In other words, we might compare the reliability of different LCDPs, and the LCDPs that get higher feedback in these items might be more reliable. User interface consistency and tool response did not show correlations with any experience item. In this study, they could not be predicted by participants' feelings.

Despite the above findings, many open questions remain and there is a need for more research to further test the validity of the questionnaire items. For instance, why do some items perform differently across tasks, and is it related to the difficulty of the task? Why did *Productive – Unproductive* show a significant correlation with the ease of understanding the task? One possibility is that participants could not distinguish between their ability to understand the task and the outcomes they produced. This could have interesting implications for studying DX in general: the memory of a development episode may be distorted by the outcomes, meaning in this case that developers determine their understanding of a task based on whether they were able to produce something tangible, regardless of whether the outcome conformed to what was intended from the outset or not. This would be consistent with prior research showing that, along with the most intense part, the final part of an experience episode has a strong influence on the assessment of the whole episode [44]. Investigating whether interventions in the final part of an episode have an impact on DX, as measured here, could be an interesting avenue for further research.

Finally, the approach taken here has aimed to examine the instrument to inform the future development of DX questionnaire instruments. To obtain instruments with better validated psychometric properties, the results and insights developed in this paper should be combined with a rigorous instrument development approach. Using factor analysis and correlating candidate instruments with existing DX and UX instruments as well as other ways of obtaining experiential information are possible ways to obtain more strongly validated DX measurement instruments.

7.5. Limitations

The main aim of our study was to develop a questionnaire with a grounding in present DX literature and expert knowledge. The primary concern was how to operationalise the DX concept in the context of LCDPs to capture important experiential factors in development episodes. Thus, we consider *construct validity* and *content validity* to be the most important criteria for the study, i.e., to what extent the questionnaire reflects the concepts it should reflect. Content and face validity have been identified as important for instrument development related to experiential characteristics in other fields [45] and we consider the perspective of experts and practitioners to be crucial for studies on DX measurement as well. Correlational analyses are commonly used to assess construct validity along with incorporating qualitative information from experts and practitioners, as we have done in this paper.

The combination of using prior literature about DX and the Delphi study with experts serves to increase the confidence that the questionnaire reflects prevailing understandings of episodic DX of LCDPs. As we have noted above, prior literature does not, to the best of our knowledge, provide a measurement instrument specifically aimed to measure episodic DX. The literature therefore served to inform the study about potential questionnaire items relevant to DX in general, while our analysis of those items against the DX conceptual framework [1], our knowledge of DX and LCDPs, and the comments from experts in the Delphi study served to refine the questionnaire to capture the desired aspects of episodic DX as well as possible. Our analysis of the task-based test results shows that the questionnaire is understandable by practitioners and can be used to distinguish between different kinds of experiences by developers with different backgrounds. While a single study is not enough to provide construct validity, we nevertheless consider the primary concern of our study to have been adequately addressed. We acknowledge that further work is needed to gain more confidence in the content and criterion validity of the instrument. However, if the questionnaire omits some aspect of the experience that would be relevant for development episodes using LCDPs, adding elements must be balanced against the practical effort required for participants to fill in the questionnaire.

Internal validity is concerned with the extent to which cause-effect relationships are trustworthy within the context of a study. In our case, selection bias and confounding could have influenced the differences in participants' responses.

Prior knowledge is a significant influencing factor when studying episodic developer experience of LCDPs. One of the goals of this study was to discover the impact of prior knowledge on using the LCDP to perform tasks. We attempted to understand whether programming expertise, LCDP expertise, and design expertise have impacts on episodic experience when participants are using LCDPs to perform tasks.

ICC analysis indicates that programming expertise does make a difference to the episodic experience of the LCDP: participants with no knowledge of programming rated the experience items consistently, while those with programming knowledge rated them differently. We observe that our measure of programming knowledge was not detailed enough to discern meaningful patterns among the latter group. Based on the participants' LCDP and design tool expertise, no consistencies were found between participants, regardless of which group they were involved. Due to the lack of a clear definition and control of participants' backgrounds, the correlation between background and experience is not significant. However, by explicitly controlling for and analysing expertise on different levels, we attempted to guard against selection bias and confounders; in other words, prior differences in expertise were accounted for in the study design. Further validation is needed to discover other potential confounders.

A further threat to internal validity is researcher bias: we could have accidentally influenced participants in the study. We took steps to guard against this. In the Delphi study, we presented experts with our in-progress questionnaire and asked them to comment without stating our own opinions. Only after they had provided their feedback did we engage in a discussion where we explained our rationale behind items. In the task sessions, the training task provided participants with an introduction to the most important functions of the LCDP, and they then had to carry out the two other tasks on their own. In cases where they asked for help, we did not immediately solve the problem for them but instead encouraged them to keep exploring until they ran out of time. Only after filling in the questionnaire did we help them complete the task if they still wished to.

Another apparent threat undoubtedly arises from the small sample, the use of only two tasks, and the use of only one LCDP, which raises concerns about *external validity*.

The study groups had one to three participants and the total number of participants (19) is not representative of any larger population of developers. However, we did not aim to validate the questionnaire in a well-defined population. Rather, we focus here on obtaining a questionnaire with good enough construct validity to warrant use in practical settings together with other means of assessment and for further research. The number and nature of tasks also prevent us from drawing far-reaching conclusions about the relationship between task traits and questionnaire items. Both diversified and repeated tasks are necessary to strengthen the validity of the questionnaire for the study of episodic experience.

Finally, the test only covers one LCDP and thus it can be questioned to what extent the results are generalisable to other LCDPs or development tools in general. However, we argue that this aspect is not relevant to the present study. The study design did not rely on any particular aspect of the LCDP used, and the questionnaire does not contain any items that would rely on the characteristics of any particular LCDP. The questionnaire specifically directs participants to report on their personal experience of the session they conducted (i.e., the development episode), not to assess the tool. Thus this should not be a primary validity concern, and we note that many widely used usability and UX instruments have only been validated through years of use and study after their initial introduction. Still, further studies could assess external validity further by varying the LCDP used.

8. Conclusions

In this study, we set out to obtain an understanding of end-user developers' episodic experience when building an application on an LCDP to guide how such experiences can be measured. To this end, we designed a questionnaire with ten experience items to measure the episodic experience of developers using LCDPs after they had completed a specific bounded task. All items were based on prior literature on the characteristics of LCDPs and developer experience. The items were refined based on expert feedback in a Delphi study. We then used a task-based test to obtain data from the actual use of the questionnaire in conjunction with an LCDP. Participants also provided background information on their prior expertise and received a short training session using the LCDP. We analysed correlations between the experience items in our questionnaire and participant assessments of tool reliability, user interface consistency, tool response, task understanding, and task completion, as well as their assessment of prior expertise.

The results show that all experience items in the questionnaire were related to the characteristics of the tool and the difficulty of the task. Perceived task difficulty, tool reliability, and task completion appear to play a role in episodic experience. We emphasise the importance and necessity of separating personal experience from the assessment of tasks and tools. In terms of the impact of developers' background, prior programming experience played an important role in measuring the episodic experience of LCDPs, but a more detailed definition and control of background is needed to verify this conclusion.

The findings in this study can be used to provide insights for further research into developer experience measurement. In particular, we advise instrument developers to develop a detailed understanding of the context of development and to carefully design their instruments for the desired time scale of experience, e.g., anticipatory, momentary, episodic, or cumulative. The questionnaire can be used, taking its limitations into account, for assessing the episodic developer experience of developers using LCDPs. We recommend

that it be used alongside other assessment methods such as think-aloud protocols or post-task interviews. We believe it could prove useful to assess development episodes involving other kinds of development tools besides LCDPs, but further research is required to validate its use for such purposes.

Acknowledgements

We would like to express our sincere thanks to the experts and study participants for their involvement in the research.

References

[1] F. Fagerholm and J. Münch, "Developer experience: Concept and definition," in *International Conference on Software and System Process (ICSSP)*. IEEE, 2012, pp. 73–77.

[2] F. Fagerholm, *Software developer experience: Case studies in lean-agile and open source environments*, Ph.D. dissertation, University of Helsinki, Faculty of Science, Department of Computer Science, Helsinki, Finland, 2015.

[3] D. Graziotin, X. Wang, and P. Abrahamsson, "Are happy developers more productive?" in *International Conference on Product Focused Software Process Improvement*. Springer, 2013, pp. 50–64.

[4] M.A. Storey, T. Zimmermann, C. Bird, J. Czerwonka, B. Murphy et al., "Towards a theory of software developer job satisfaction and perceived productivity," *IEEE Transactions on Software Engineering*, Vol. 47, No. 10, 2019, pp. 2125–2142.

[5] N. Baddoo, T. Hall, and D. Jagielska, "Software developer motivation in a high maturity company: a case study," *Software process: improvement and practice*, Vol. 11, No. 3, 2006, pp. 219–228.

[6] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "What happens when software developers are (un)happy," *Journal of Systems and Software*, Vol. 140, 2018, pp. 32–47.

[7] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-user development: An emerging paradigm," in *End user development*. Springer, 2006, pp. 1–8.

[8] Y. Luo, P. Liang, C. Wang, M. Shahin, and J. Zhan, "Characteristics and challenges of low-code development: The practitioners' perspective," in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 2021, pp. 1–11.

[9] M. Tisi, J.M. Mottu, D.S. Kolovos, J. De Lara, E.M. Guerra et al., "Lowcomote: Training the next generation of experts in scalable low-code engineering platforms," in *STAF 2019 Co-Located Events Joint Proceedings: 1st Junior Researcher Community Event, 2nd International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems, and 1st Research Project Showcase Workshop co-located with Software Technologies: Applications and Foundations (STAF 2019)*, 2019.

[10] J. Cabot, "Positioning of the low-code movement within the field of model-driven engineering," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20. ACM, 2020, pp. 1–3.

[11] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio, "Supporting the understanding and comparison of low-code development platforms," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2020, pp. 171–178.

[12] F. Khorram, J.M. Mottu, and G. Sunyé, "Challenges and opportunities in low-code testing," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20. ACM, 2020, pp. 1–10.

[13] H. Lee and Y. Pan, "Evaluation of the nomological validity of cognitive, emotional, and behavioral factors for the measurement of developer experience," *Applied Sciences*, Vol. 11, No. 17, 2021, p. 7805.
[14] E. Karapanos, J. Zimmerman, J. Forlizzi, and J.B. Martens, "User experience over time: an initial framework," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2009, pp. 729–738.
[15] B. Callary, P. Werthner, and P. Trudel, "How meaningful episodic experiences influence the process of becoming an experienced coach," *Qualitative research in sport, exercise and health*, Vol. 4, No. 3, 2012, pp. 420–438.
[16] P. Marti and I. Iacono, "Anticipated, momentary, episodic, remembered: The many facets of user experience," in *Federated Conference on Computer Science and Information Systems (FEDCSIS)*. IEEE, 2016, pp. 1647–1655.
[17] M. Greiler, M.A. Storey, and A. Noda, "An actionable framework for understanding and improving developer experience," *IEEE Transactions on Software Engineering*, 2022.
[18] K. Kuusinen, "Are software developers just users of development tools? assessing developer experience of a graphical user interface designer," in *Human-Centered and Error-Resilient Systems Development*. Springer, 2016, pp. 215–233.
[19] K. Kuusinen, "Software developers as users: Developer experience of a cross-platform integrated development environment," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2015, pp. 546–552.
[20] T. Kaltio and A. Kinnula, "Deploying the defined SW process," *Software Process: Improvement and Practice*, Vol. 5, No. 1, 2000, pp. 65–83.
[21] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "On the unhappiness of software developers," in *Proceedings of the 21st international conference on evaluation and assessment in software engineering*, 2017, pp. 324–333.
[22] K.R. Linberg, "Software developer perceptions about software project failure: A case study," *Journal of Systems and Software*, Vol. 49, No. 2–3, 1999, pp. 177–192.
[23] A. Bobkowska, "On explaining intuitiveness of software engineering techniques with user experience concepts," in *Proceedings of the International Conference on Multimedia, Interaction, Design and Innovation*, 2013, pp. 1–8.
[24] F. Fagerholm, M. Felderer, D. Fucci, M. Unterkalmsteiner, B. Marculescu et al., "Cognition in software engineering: A taxonomy and survey of a half-century of research," *ACM Comput. Surv.*, Vol. 54, No. 11s, 2022.
[25] M. Sánchez-Gordón and R. Colomo-Palacios, "Taking the emotional pulse of software engineering – A systematic literature review of empirical studies," *Information and Software Technology*, Vol. 115, 2019, pp. 23–43.
[26] G. Fischer, D. Fogli, and A. Piccinno, "Revisiting and broadening the meta-design framework for end-user development," in *New perspectives in end-user development*. Springer, 2017, pp. 61–97.
[27] B.R. Barricelli, F. Cassano, D. Fogli, and A. Piccinno, "End-user development, end-user programming and end-user software engineering: A systematic mapping study," *Journal of Systems and Software*, Vol. 149, 2019, pp. 101–137.
[28] W.W. Cotterman and K. Kumar, "User cube: A taxonomy of end users," *Communications of the ACM*, Vol. 32, No. 11, 1989, pp. 1313–1320.
[29] M. Hirzel, "Low-code programming models," *Communications of the ACM*, Vol. 66, No. 10, 2023, pp. 76–85.
[30] C. Richardson, J.R. Rymer, C. Mines, A. Cullen, and D. Whittaker, "New development platforms emerge for customer-facing applications," Forrester, Cambridge MA USA 15, Tech. Rep. 16, Jun 2014.
[31] R. Waszkowski, "Low-code platform for automating business processes in manufacturing," *IFAC-PapersOnLine*, Vol. 52, No. 10, 2019, pp. 376–381.
[32] N. Dalkey and O. Helmer, "An experimental application of the Delphi method to the use of experts," *Management Science*, Vol. 9, No. 3, 1963, pp. 458–467.

[33] A. Nylund, "A multivocal literature review on developer experience," Master's thesis, Aalto University School of Science, 2020. [Online]. https://aaltodoc.aalto.fi/items/d822c160-0cc6-41d1-9c29-72470ec2ad13

[34] S.A. Jackson, A.J. Martin, and R.C. Eklund, "Long and short measures of flow: The construct validity of the FSS-2, DFS-2, and new brief counterparts," *Journal of Sport and Exercise Psychology*, Vol. 30, No. 5, 2008, pp. 561–587.

[35] R.M. Ryan, "Control and information in the intrapersonal sphere: An extension of cognitive evaluation theory." *Journal of Personality and Social Psychology*, Vol. 43, No. 3, 1982, p. 450.

[36] M. Hassenzahl, S. Diefenbach, and A. Göritz, "Needs, affect, and interactive products–facets of user experience," *Interacting with Computers*, Vol. 22, No. 5, 2010, pp. 353–362.

[37] D. Alwin, "Feeling thermometers versus 7-point scales: Which are better?" *Sociological Methods and Research*, Vol. 25, No. 3, 1997, pp. 318–340.

[38] H. Taherdoost, "What is the best response scale for survey and questionnaire design; Review of different lengths of rating scale/attitude scale/likert scale," *International Journal of Academic Research in Management*, Vol. 8, No. 1, 2019, pp. 1–10.

[39] D. Gao and F. Fagerholm, "Supplementary materials for the paper 'Measuring end-user developers' episodic experience of a low-code development platform – A preliminary study'," 2023. [Online]. https://doi.org/10.5281/zenodo.7515833

[40] M. Lynn, "Determination and quantification of content validity," *Nursing Research*, Vol. 35, 1986, pp. 382–386.

[41] D.V. Cicchetti, "Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology." *Psychological Assessment*, Vol. 6, No. 4, 1994, p. 284.

[42] T.K. Koo and M.Y. Li, "A guideline of selecting and reporting intraclass correlation coefficients for reliability research," *Journal of Chiropractic Medicine*, Vol. 15, No. 2, 2016, pp. 155–163.

[43] B. Laugwitz, T. Held, and M. Schrepp, "Construction and evaluation of a user experience questionnaire," in *Symposium of the Austrian HCI and usability engineering group*. Springer, 2008, pp. 63–76.

[44] D.A. Redelmeier and D. Kahneman, "Patients' memories of painful medical treatments: real-time and retrospective evaluations of two minimally invasive procedures," *Pain*, Vol. 66, No. 1, 1996, pp. 3–8.

[45] J. Connell, J. Carlton, A. Grundy, E. Taylor Buck, A.D. Keetharuth et al., "The importance of content and face validity in instrument development: Lessons learnt from service users when developing the recovering quality of life measure (ReQoL)," *Quality of Life Research*, Vol. 27, 2018, pp. 1893–1902.

e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240106, DOI: 10.37190/e-Inf240106

Activity-Based Detection of (Anti-)Patterns: An Embedded Case Study of the Fire Drill

Sebastian Hönel*, Petr Picha**, Morgan Ericsson***, Premek Brada****, Welf Löwe*****, Anna Wingkvist*****

*Faculty of Technology, Department of Computer Science and Media Technology, Linnaeus University, Sweden

**Faculty of Applied Sciences, Department of Computer Science and Engineering, University of Western Bohemia, Czechia

***Faculty of Technology, Department of Computer Science and Media Technology, Linnaeus University, Sweden

****Faculty of Applied Sciences, Department of Computer Science and Engineering, University of Western Bohemia, Czechia

*****Faculty of Technology, Department of Computer Science and Media Technology, Linnaeus University, Sweden

sebastian.honel@lnu.se, ppicha@ntis.zcu.cz, morgan.ericsson@lnu.se, brada@kiv.zcu.cz, welf.lowe@lnu.se, anna.wingkvist@lnu.se

"While it is certainly useful to study the successful ways people solve problems, the old adage that we learn from our mistakes suggests that studying failures might be even more fruitful.
– Neill et al.

Abstract

Background: Nowadays, expensive, error-prone, expert-based evaluations are needed to identify and assess software process anti-patterns. Process artifacts cannot be automatically used to quantitatively analyze and train prediction models without exact ground truth.
Aim: Develop a replicable methodology for organizational learning from process (anti-)patterns, demonstrating the mining of reliable ground truth and exploitation of process artifacts.
Method: We conduct an embedded case study to find manifestations of the Fire Drill anti-pattern in $n = 15$ projects. To ensure quality, three human experts agree. Their evaluation and the process' artifacts are utilized to establish a quantitative understanding and train a prediction model.
Results: Qualitative review shows many project issues. (i) Expert assessments consistently provide credible ground truth. (ii) Fire Drill phenomenological descriptions match project activity time (for example, development). (iii) Regression models trained on ≈ 12–25 examples are sufficiently stable.
Conclusion: The approach is data source-independent (source code or issue-tracking). It allows leveraging process artifacts for establishing additional phenomenon knowledge and training robust predictive models. The results indicate the aptness of the methodology for the identification of the Fire Drill and similar anti-pattern instances modeled using activities. Such identification could be used in post mortem process analysis supporting organizational learning for improving processes.

Keywords: Anti-Patterns, Fire-Drill, Case-study

Submitted: 3 Mar. 2023; Revised: 4 Jan. 2024; Accepted: 26 Feb. 2024; Available online: 1 Mar. 2024

1. Introduction

A pattern describes a reoccurring problem and its prototypical solution [2]. If the solution exacerbates – rather than ameliorates – a problem, a pattern becomes an *anti-pattern* [3, 4]. There exist anti-patterns related to, e.g., software architecture, that have a palpable effect on product quality. Other types of anti-patterns that have negative repercussions on the *process* and the resulting product could be attributed to management malpractices [5, 6]. Nelson summarizes a number of infamous project failure examples, most of which are the result of ineffective project management practices. The overall project failure categories are people, technology, product, and process [7], of which we focus on the latter. The software process is characterized by certain *activities*, such as development (e.g., implementation of features and fixing of bugs) or requirements engineering. The quality of this process depends, among other factors, on an adequate allocation of time for carrying out each activity. The quality of a software product was demonstrated to be tied closely to the quality of the software process [8]. Halvorsen and Conradi [9] even suggest the causal relation Quality(*Process*) $\Rightarrow$ Quality(*Product*). Therefore, it is worth studying phenomena related to process quality, in order to understand and improve product quality. Project failure is a frequently embraced opportunity for post mortem organizational learning [10]. Failure can often be attributed to process anti-patterns, of which the so-called "Fire Drill" is a prominent example, due to its clearly discernible symptoms [11, 12]. It is often characterized by "months of monotony [...]" (unsatisfactory early project progress) that is "[...] followed by a crisis" [4], due to forcing immediate delivery [13].

The Fire Drill is one of many existing management anti-patterns known to exacerbate software processes fraught with problems [14]. Usage of such anti-patterns for organizational learning is inhibited by a variety of factors today. The most pronounced problem is perhaps the lack of a quantitative description: anti-patterns are only described qualitatively using, e.g., structured templates with elements such as causes, symptoms, or consequences [5]. This effectively constrains the available methods of analysis to qualitative assessments. The manual evaluation depends on experts knowledgeable and available in the problem domain [15]. The problem is further aggravated by the fact that qualitative assessment is labor-intensive and error-prone because experts are likely to introduce their own subjective bias [16]. Although software development processes produce a multitude of diverse digital artifacts either as a by-product (e.g., source code) or based on the use of project or application lifecycle management (ALM) tools, most of these data cannot facilitate anti-pattern instance detection [17]. Even though there exist approaches that take advantage of such artifacts by using more formal and technical models, or thresholds and rules, the scope of their utility is severely limited to single process aspects, such as progress, variable dependencies, or estimating uncertainty, e.g., [18–22]. Lastly, scarcely available historical data, that is, data on past projects, impede or prohibit comprehensive and generalizable learning that results in organizational knowledge.

Many of the existing approaches have *software process improvement* as the ultimate goal. They first establish some *white-box* model and then analyze the effect of the measures implemented within the model boundaries [23, 24]. In contrast, we first conduct a longitudinal embedded case study [25, 26], which qualitatively evaluates the presence and severity (i.e., how strongly the phenomenon manifests) of a Fire Drill in $n = 15$ student projects collected over a period of three years, to find an accurate ground truth[1]. Through observer and data triangulation [29], as well as inter-rater reliability assessment [30], we ensure

[1]The feasibility of studies under similar constraints of scarce data have previously been successfully conducted in, e.g., biomedical engineering [27] and material sciences [28].

a minimum quality of the observed evidence that makes the foundation for the ground truth as is required for further analyses [31]. The ground truth is then used to leverage previously unusable artifacts to establish quantitative measures and to derive knowledge. An adaptive training of a gray-/black-box model allows predicting the (severity of the) anti-pattern instances using the implemented measures [32]. Finally, the new insights are propagated back to the studied case. From the digital project artifacts, we can confidently derive the carried-out activities, such as adding features or engineering of requirements[2]. Then, the Fire Drill is modeled in terms of time spent on (temporal accumulations of) these activities. We learn that the phenomenon is sensitive to a certain balance of these activities, that is, how much time on each activity is spent in relation to any other activity, at any point in time. We argue that this is also the case for many related or similar phenomena. Therefore, next to the immediate contributions, the intended main contribution is the methodology that will allow full or partial replication of this study (using, e.g., different phenomena or modified contexts).

1.1. Data used in the study

Embedded case studies, such as the present one, are characterized by and rely on qualitative and quantitative data [35]. The study design is split to appropriately study either type of data. For the remainder of this work, we refer to these as *type-I data* and *type-II data*, respectively. Most data are produced in the development process and related to project planning, application lifecycle management, version control, and documentation. As either type contains both qualitative and quantitative artifacts, we define the types of data as follows.

Type-I Data. In addition to archival records, this case study uses direct observations, memory logs, meeting minutes (e.g., from stand-ups, retrospectives, customer meetings, or iteration planning), participant observation, team experience reports, customer comments, mentors' assessment notes, and wikis (or other types of note collections) maintained by each team as primary data sources for the qualitative portion. These data are largely unstructured and were recorded mostly subjectively. The type-I archival data is treated as a set of distinct data sources; the production and extraction of information related to these sources are outside the scope of the research method in this study [29].

Type-II Data. For the quantitative part, this study mainly uses out-of-sample testing for performing model validation and generalization error estimation. The data is structured and comes from the archives of the ALM tools. It includes version control systems that hold the source code (and its commits) and raw data from the underlying issue-tracking tools. The latter are mostly tickets that have been assigned a category and time estimates. Although there is some uncertainty in these tickets, all other data used in the quantitative analysis were objectively recorded.

1.2. Objective

Clearly defined objectives are a common element of the research design for case studies [29]. We pursue a single, principal objective refined into research questions to which we offer some answers. For this case study, the principal objective is defined as follows. *Automate the post mortem Fire Drill severity assessment, by utilizing the qualitatively won ground*

[2]There is a partial semantic overlap between our activities and the so-called disciplines as used in the (Rational) Unified Process [33, 34].

truth and selected suitable quantitative, objective type-II artifacts. Pursuing this goal will necessarily result in the extraction and generation of knowledge from the type-II artifacts. The point of departure for the principal objective is one in which, before conducting the case study, we are relatively certain that the Fire Drill, given its existing phenomenological descriptions, is detectable and that some projects will exhibit one. Although this was the result of our pilot study (see Subsection 3.2), we are yet uncertain of the quality of the qualitative evaluation (whether severity can be determined *sufficiently* accurately). Although all data were available at the beginning of the study, that is, type-I and type-II data, we did not know if and how many of the type-II artifacts would be suitable for automated detection, implemented as a regression model. Also, manual facilitation of the type-II data is difficult, as the Fire Drill was not previously described from a quantitative perspective. Manual analysis of quantitative data is further inhibited by the data size (data points and dimensionality) and possibly non-linear correlations.

1.3. Propositions, hypotheses, research questions

In this section, we will guide the reader through the main aspects of the case by presenting three sets of propositions, hypotheses, and research questions in a consecutive manner. For case study research, it is suggested to align the methodology close to these elements (however, not necessarily in this order, e.g., [29, 36]). The If-Then-styled propositions first give potential implications, while the generated hypotheses provide structure and detail to the formulated research questions.

1.3.1. Understand the Fire Drill manifestation

The first set of propositions, hypotheses, and research questions concerns the manifestation of the Fire Drill in our context directly.

Pr. 1.1: If the Fire Drill phenomenon is described well enough and present in the projects, then manifestations of it can be found using type-I data only.

Pr. 1.2: If there is agreement between independent raters, then the existing phenomenological descriptions of the Fire Drill are sufficient for a qualitative post mortem evaluation.

Pr. 1.3: If there is agreement on the absence of the Fire Drill in a project, then the evidence that is counter-indicative of the phenomenon (true negatives) can be gathered.

Hyp. 1.1: The type and quality of the type-I data allow for accurate assessment of the severity of the Fire Drill using qualitative evaluation.

Hyp. 1.2: Projects not affected by the Fire Drill can be used to derive common symptoms whose presence can indicate its absence.

RQ 1.1: Can the severity of the Fire Drill be accurately determined?

RQ 1.2: What is the nature of the Fire Drill manifestation in each of the student projects?

RQ 1.3: What evidence can be gathered indicating the absence of a Fire Drill?

1.3.2. Establish an understanding using qualitative data

The won ground truth now allows us to access, reason about, and leverage available quantitative data in an unprecedented way. The second set of propositions, hypotheses, and research questions was designed to establish a new quantitative understanding. Our approach here is characterized by instrumental motivation and nomothetic evaluation. Sets two and three facilitate quantitative project data of the same structure across all

projects. While the focus is on the sub-unit of analysis, the projects are examined together (nomothetically), instead of individually.

In the following, *importance* refers to the concept (and technique) of (determining) variable importance for the prediction of a dependent variable (here: severity) [37]. When the timeline of a project is subdivided into phases, then each feature (variable) is exclusively assigned to a single phase. Therefore, variable importance allows us to determine and compare the relative importance of each project phase by aggregating the importance of each feature.

Pr. 2.1: If a ground truth can be determined accurately enough, then it can be exploited for the analysis, interpretation, understanding, and comprehension of the quantitative data.

Hyp. 2.1: Activities closely related to those described by the Fire Drill will display characteristic behavior that is in accordance with the Fire Drill's phenomenological descriptions.

Hyp. 2.2: All project phases will exhibit non-zero importance, with the later phases being of greater importance (the Fire Drill, supposedly, is more critical towards the project end).

RQ 2.1: What are typical accumulations of (maintenance) activities characteristic of a Fire Drill?

RQ 2.2: What phases and activities are most important for predicting the presence or severity?

1.3.3. Obtain a robust predictive model

The last set of propositions, hypotheses, and research questions was designed primarily to achieve the main objective (see Subsection 1.2), namely to automate the post mortem severity assessment. As a by-product, we will also improve our quantitative understanding, as continued from the previous set. In the following, *stability* refers to two criteria simultaneously, the first of which is the expected generalization error and the second of which is the confidence interval regarding the generalization error of a predictive model.

Pr. 3.1: If any type-II data (artifacts) are suitable, training should converge toward stability with increasing amounts of training data.

Pr. 3.2: If a stable model with acceptable generalization error can be obtained, then assessing the severity of the phenomenon with it could be instantaneous.

Hyp. 3.1: Type-I ground truth and type-II data from a few projects are apt for training a regression model with acceptable stability.

Hyp. 3.2: Using either source code or issue-tracking data should yield predictive models of similar stability, as the former is more objective, while the latter is more closely aligned with the Fire Drill's described activities.

RQ 3.1 What source of data, issue-tracking or source code, yields better models?

RQ 3.2: How many data points are required for obtaining a stable predictive model?

1.4. Notions and abbreviations

Activities use an all-caps sans serif font and are abbreviated by a few letters. For example, the activity *development* is denoted by DEV. It is also used as a subscript for functions over time, that is, f_{DEV}. Similarly, empirical observations, symptoms, and consequences

use few letters and a number, in typewriter font. For example, `ESC01` is the first empirical observation of a symptom/consequence.

1.5. Structure of this article

The remainder of this article is structured as follows. The next Section, 2, introduces related and relevant work. Section 3 provides background information on phenomena described using a pattern language, with a dedicated focus on anti-patterns and the Fire Drill, as well as details about the preceding pilot study. The entire design of this study and the underlying methodology is presented in Section 4. It is followed by Section 5, which is dedicated to the presentation of the results of the analyses. The validity of our study, the limitations of its results, their replicability and generalizability, as well as a summary of the results obtained and how they relate to the established propositions and hypotheses, are discussed in Section 6. The conclusion and synthesis of all results related to the case studied, as well as prospects for future work, are given in Subsection 7.1. At the end of the article, the additional appendices A through E, providing supplementary details and insights, can be found.

2. Related work

Pattern-like phenomena related to management are not models because they only describe a well-known solution to some recurring problem [3], but effectively lack efficient operationalizability (in terms of, e.g., a predictive model for presence and/or severity). Simeckova et al. [5] have identified a wide semantic gap between the qualitative (textual, unstructured, and often ambiguous) and quantitative description of pattern-like phenomena, where the latter is practically impossible to specify. This is because patterns are deliberately left abstract (or even "vague", as Alexander et al. [2] put it). Therefore, a quantitative description would require, for example, context-specific thresholds or rules according to Simeckova et al.

Currently, no software process improvement model that could adequately represent or evaluate the presence of a Fire Drill based on quantitative data exists. Brown et al. [11] started to characterize the Fire Drill anecdotally, describing the problem it portrays, together with a refactored solution called *sheltering* (see Subsection 3.1). They did not attempt to abstract from this, i.e., there is no list of symptoms, for example. The only possible form of operationalization would be a manual evaluation of whether and to what degree the own project matches their description. Although they do not use the example of a Fire Drill, Laplante and Neill [4] were the first to collect various managerial and cultural anti-patterns and to represent each in a common, structured template. It included, for example, the anti-pattern's central concept, its dysfunction, a short vignette, a plain explanation of the anti-pattern, clues to alleviate the fallout, and – most importantly – a yes/no checklist with symptoms for identifying the anti-pattern's presence. More recently, Picha and Brada [18] started to collect and consolidate anti-patterns in a common template. They extract and gather various characteristics from each anti-pattern and translate some of their peculiarities into actionable symptoms, consequences, and solutions.

Attempts exist to operationalize various (anti-)patterns for software process quality. For example, next to the informal descriptions, there are more formal models, such as Bayesian (Belief) networks, ontologies, social networks, and Design structure matrices [6, 19–21]. Of these, only Bayesian networks are actionable to a limited extent, as they allow for modeling conditional probability distributions and visualization of dependencies between variables.

The other methods are rather concerned with encoding project management knowledge into machine-readable form. The actionability in Bayesian networks is limited to assessing uncertainty; that is, it is not suitable for detecting the presence of anti-patterns but rather a tool for exploring relations [22]. However, they may be used to measure certain aspects of the process, such as its progress or quality.

There is plenty of other research on quantifying software quality using ALM artifacts. For example, Draheim and Pekacki [38] focuses on developers' activity throughout the project using collaboration, productivity, and evolution metrics. Ramsauer et al. [39] deal with estimating the maintenance costs in software development, and Tamburri et al. [40] study the organizational aspects of software projects and communities. Talpová and Čtvrtníková [41] use ALM data to investigate Scrum anti-patterns in a case study, though only as a secondary source of information combined with surveys and interviews. Hachemi [42] explores the reuse of patterns (in a more colloquial sense) in the modeling of software development processes. Although other researchers also focus on (anti-)patterns, they do not use ALM data. Frtala and Vranic [43] research organizational patterns and their adoption in individual organizations and projects using gamified learning techniques. Settas and Stamelos [20], as well as Stamelos [6] aim specifically at project management anti-patterns, their knowledge base, and effective communication using heterogeneity of personalities and character traits of developers. A major part of the research efforts also deals with the modeling of (anti-)patterns and detection through languages and ontologies [44], or models like the software process engineering meta-model [5] and the business process model and notation [45].

To the best of our knowledge, no one has previously attempted to operationalize an anti-pattern using the approach presented in this study. It was previously shown that performing post mortems is a viable path to organizational learning [10] and that learning from anti-patterns is deemed a way to eventually master management knowledge [6]. Also, it appears that examining and learning from eventuated anti-patterns is not limited to the context of software development. For example, Awad et al. [46] use them to detect compliance violations of business processes. Lastly, the Fire Drill is a phenomenon that can lead to anything between ever-so-slight and severe, negative repercussions, such as total project failure. Studying project failure typically results in the discovery of many interrelated symptoms and consequences [47]. Although, compared to the application of common software process improvement models, we address only a single quality goal, we observe numerous different symptoms and consequences. It appears that the statement of Neill et al. [1] about the fruitfulness of studying project failure has become true.

3. Background

This section provides some background information about phenomena described using a pattern language, with a focus on the Fire Drill anti-pattern. It also gives an overview of previous work, that served as a pilot study.

3.1. Phenomena described using a pattern language

Generally, patterns are reoccurring and identifiable phenomena [2], that are especially prevalent in phases of design, project management, and in software development processes [48]. A pattern provides a general and proven solution to a common problem.

> *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."* – Alexander et al.

However, as the definition by Alexander et al. [2] implies, the descriptions are deliberately left vague in order to leave room for case-specific applications of the solution the pattern represents. The definition also highlights the challenge of patterns to date, which lies in objectively definable and quantifiable problems, and the lack of technical and automated solutions to these. While efforts for a more technical way of describing patterns have been undertaken (e.g., [1, 11, 18]), they are predominantly described using a *pattern language*, structured text, or a template [6]. Therefore, patterns are most often not described quantitatively and lack clear connections to quantifiable properties, such as software metrics.

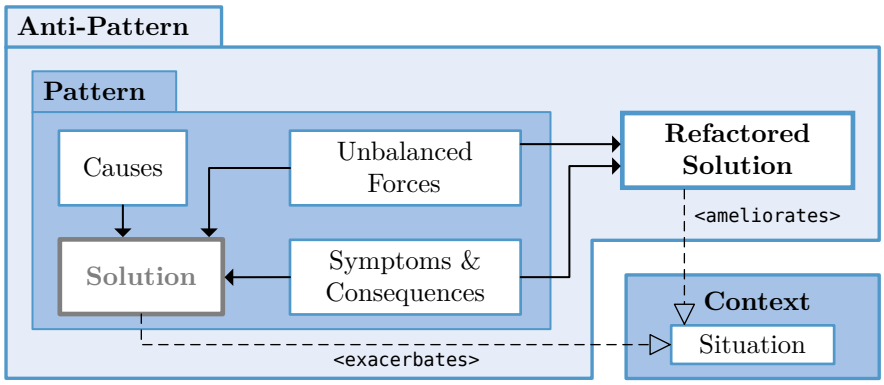


Figure 1. Some common elements of patterns and anti-patterns. The context element refers to the context (and its problematic situation) of the case study itself (see Subsection 4.1)

An **anti-pattern** can be conveyed as a larger concept than a pattern, in a way that it encompasses the elements of a pattern, but also adds an additional, so-called *refactored* solution [4]. The refactored solution is, for example, a restructured, improved, or otherwise optimized version of the original (regular) solution. The regular solution that comes from the enclosed pattern is what creates the anti-pattern in the first place because its application constitutes malpractice that **exacerbates** the problem, rather than ameliorating it (see Figure 1). Therefore, the actual mending solution is the refactored solution, rather than the regular solution. The refactored solution is an element that is known for some anti-patterns, but not for all. If present, it is commonly included in the pattern language or structured template (e.g., [14]). As a result, anti-pattern project management phenomena pose threats to project quality and delivery and are the outcome of human error. The ramifications include but are not limited to, developer churn, interpersonal and organizational tensions, a product of poor quality, delayed delivery, or even total project failure. The notion of an anti-pattern is, therefore, closely related to that of *project risk* [6].

3.1.1. The Fire Drill

The presence of a Fire Drill is a kind of problematic software project situation. Most recently, the Fire Drill was described following a common, structured template (see Appendix A). Rudimentarily speaking, it is construed as a short, desperate, and highly active phase towards the end of a software development project. It is preceded by a considerably longer,

much less productive phase. Often, a disproportionately large amount of the project's resources is spent in the first phase, without advancing the project at the required pace. When management acknowledges the urgency of the imminent due date, the start of the typical Fire Drill is heralded [3, 11, 12].

While the Fire Drill is a phenomenon that unfolds as the result of poor management during the development phase of a project, some projects are predestined to exhibit one due to poor time allocation for the preceding approval and budgeting processes. This phenomenon is known as the "fuzzy front end" and often results in an aggressive development schedule from the beginning [49].

3.1.2. Patterns related and similar to the Fire Drill

The Fire Drill is related to other phenomena, some of which it is partially indistinguishable from. This happens because a certain set of indicators and symptoms and consequences are similarly indicative of other phenomena. Some related anti-patterns are, for example, "Analysis Paralysis"[3] (a potential cause) or "Collective Procrastination" [18] (a more generic case). Other anti-patterns, such as "Half Done is Enough"[4], or "Brook's Law" [50] may constitute typical symptoms associated with an early-/late-stage Fire Drill. The anti-pattern "Cart Before the Horse" emerged as part of a Fire Drill in some of the affected projects. While it is a pattern of its own, it is a typical, severe, and frequently occurring symptom of a Fire Drill (see `ESC2` and `E20` in Appendix C), that may have high project risk as one of its consequences. What is furthermore similar, is that these phenomena, in theory, can also be identified based on the ALM data or ongoing activities.

3.2. Previous work

In an earlier paper that served as a *pilot study* [51] for this work, we primarily investigated the type-II data. Next to exploring and visualizing the work carried out in the projects, the goal was to assess whether a model or a plain decision rule for presence detection can be derived from the data and applied to future projects. That study did not include a qualitative evaluation of the Fire Drill in each of the projects. Furthermore, the absence of more than two raters prohibited a proper assessment of the inter-rater agreement and the quality of their findings. From the pilot study, we conclude that the usage of naive models, whether expert-designed or purely data-driven, is not beneficial, as a model requires a more adequate representation of its features. We attempt to represent the time spent on activities in two different ways. After exploring the data, we explicitly define three activities for issue-tracking that are related to requirements engineering, development, and descoping (see Subsection 4.5.1). For source code data, we predict the so-called maintenance activity [52] that is associated with each commit [53] (see Subsection 4.5.2). These activities are related to adding features, correcting faults, and perfective changes (e.g., maintenance). For each instance of an activity, it is always recorded *when* it happened. This allows us to detect temporal accumulations of comparatively lower and higher density. In issue-tracking, we additionally have access to the duration (i.e., how *much* time was spent) of each recorded activity. The duration for commits, however, remains unknown. For issue-tracking data, we chose a cumulative and normalized representation, since this data tends to be more

[3] Analysis Paralysis. 2017. http://wiki.c2.com/?AnalysisParalysis

[4] Half Done Is Enough. 2023. http://wiki.c2.com/?HalfDoneIsEnough

scarce (for example, bug tickets are only opened rarely). For source code data, we choose to represent the time spent as continuous-time random variables. We conclude that the latter representation is the most suitable for either type of data. Furthermore, we suggest using weighted density estimation for issue-tracking data, additionally considering the time spent as a weight for the temporal accumulations of these activities. We recommend using continuous probability densities also for another reason:It is straightforward to derive two kinds of features from them. First, a density can be integrated along an interval for estimating the relative amount that was carried out for an activity. Second, calculating a divergence between two densities is well understood and can be exploited for identifying disparities between activities. It is likely that common regression models (mind they require an accurate ground truth) will outperform the pilot study's approach using our findings. While a binary decision rule for presence detection can achieve a respectable accuracy, it is of obviously limited use as a severity assessment device and prone to producing false positives or negatives. Attempts to create a more fine-nuanced rule failed.

4. Case study design

We perform a *single-case embedded* case study based on *intrinsic* and *instrumental* motivation. We use qualitative and quantitative data and present the results in a mostly structured format [26]. From Sjøberg [54, 55] we may understand "a case [...] as a single, empirical configuration of actors, activities, technologies, and artifacts, all within a context." However, while all projects *do* share the same case, their empirical configuration varies. This circumstance necessitates the application of an embedded design. For example, in each project, a different product is developed, by a different group of students, applying individual practices (to some degree) to achieve their goals (see Appendix B for the full project setup). Therefore, the multiplicity rather lies in the analysis units (the projects themselves) and not in the cases, requiring an embedded design. A non-embedded design, if not studying multiple cases, would be concerned with a single unit of analysis and, therefore, not be a suitable choice for this work. Embedded case studies propagate the findings from the analyzed units back to the single case studied. Yin [25] notes that the project-level data may be highly quantitative, and the original evaluation would become a project study, i.e., a multiple case study of different projects if there is no investigation at the level of the original case. Scholz and Tietje [26] note that the multiplicity of evidence in an embedded case study is investigated partly in the projects, each focusing on different and salient aspects of the original case. In a multiple case study, each case should serve a specific purpose within the general scope of the investigation, which does not apply here. Furthermore, all propositions, hypotheses, and research questions are the same across all embedded units. Therefore, we only have a single case of study [25].

Multiple case studies follow a replication logic that starts with uncovering a significant finding that is subsequently replicated using additional case studies. However, our study is a single evaluation (concerning a single case) across multiple projects. The logical sub-units (or embedded units) are the selected $n = 15$ projects, which makes a holistic design inapplicable and warrants an embedded design instead. It is common for embedded case studies to facilitate (sampling of) quantitative data and the application of statistical analyses [35]. The **case** (subject, or main unit of analysis) is *the Fire Drill within a software engineering course* (see Figure 2). The course is the *Advanced Software Engineering course*

at master level[5] conducted during the second out of a four-semester study at the University of West Bohemia in Pilsen. Given the setup of the projects (e.g., agile, iterative, milestones, etc.), they are naturally subject to the Fire Drill phenomenon. The full context of our study is given in Subsection 4.1.

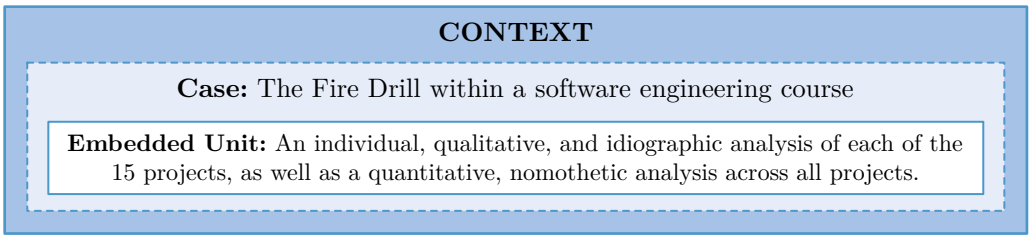


Figure 2. Case study design

Since we perform an embedded case study in the realm of software engineering, our method is closely aligned with guidelines from Runeson et al. [29] and Wohlin et al. [56]. The research strategy chosen here is exploratory, explanatory, and of an improving nature. It is not descriptive, as we are not trying to portray the current state of the Fire Drill phenomenon [29] (for that, the interested reader is referred to Subsection 3.1.1, Appendix A, and [3, 11]). It is, however, exploratory and explanatory since we seek new insights into the phenomenon's manifestation within the chosen context. The case is studied to interpret and explain symptoms, causes, and consequences and establish a connection to quantitative data that cannot be fully utilized. This is closely connected with the improving nature of this study as we attempt to add to the current qualitative understanding and establish a new quantitative awareness.

Wohlin and Rainer [55] provide a case study checklist. The first requirement is an identifiable case, which we have presented here. The second requirement is that of a real-life context. It is satisfied since we do not attempt to generalize beyond the chosen context (e.g., industrial). The third requirement is that of using multiple data collection methods. The variety of methods and sources used is described in Subsection 1.1. The fourth requirement is the study of a contemporary phenomenon. It is satisfied by us studying the projects as they occur. The last requirement is that the researchers do not act as change agents in the projects as they unfold. Although one of the raters was a mentor in most of the projects, that role was purely passive with respect to at least the Fire Drill phenomenon (see Subsection 4.1.3). This is evident because some projects showed strong manifestations of the phenomenon (or other anti-patterns) regardless. Furthermore, in software engineering, case studies are expected to establish long-term objective(s) (as opposed to action research that favors short-term change), which we did (see Subsection 1.3). We came to the characterization of our study as a single-case embedded case study, together with these properties, after discussing its nature and virtues in detail [57]. Although our analysis for the first part happened post mortem, we do not necessarily apply the full spectrum of software project post mortems. Therefore, we must separate ourselves from post mortems and processes as defined in, e.g., [58, 59]. Although our analysis partially resembles what Tiedeman describes as postmortem planning and design/verification, their conceptual framework is not applicable here for the above reasons.

[5]Course homepage. 2023. https://courseware.zcu.cz/portal/studium/courseware/kiv/aswi/?pc_lang=en

Case studies are *"[…] studies of something general, and of something particular […]"* [60]. These two different moments are characterized by an *idiographic* and a *nomothetic* part. Therefore, this study is divided into two consecutive parts (see Subsections 4.4 and 4.5). The first qualitative part seeks to gain an understanding of the Fire Drill's manifestation in student projects. Here, we make an individual, idiographic effort to understand the phenomenon in its context. This is also where we assign a severity to each project (the ground truth). The first part is closely aligned with the first set of propositions, hypotheses, and research questions as presented in Subsection 1.3. The motivation here is mainly of what Scholz and Tietje [26] call *intrinsic* character, meaning that we have a somewhat personal incentive to understand this case, as one of the authors has previously been involved in the conduct of these projects and will continue to be in the future.

4.1. Context

We report the context of this study aligned with the applicable points of the framework, as suggested by Petersen and Wohlin [61] and Runeson et al. [29]. Specifically, we report details on the products (what was manufactured in each project), processes (activities, artifacts), practices (tools and techniques), people (students and researchers), organizations (the university), and the (hypothetical) market (the customer). Appendix B lists, for each project, what kind of application was produced, its size, the used programming language(s), the number of team members, project duration in days, number of iterations and issues, time logged on issue-tracking activities, and number of commits per maintenance activity.

4.1.1. Product

Each team developed a single product over the course period, commissioned by the customer. Typically, an individual customer was assigned to each project. It is up to the customer's discretion to select an appropriate balance between product maturity and quality and the number of features. Typically, two-week iterations were used to ensure a certain minimum level of product completeness, quality, and installability.

4.1.2. Processes and practices

The goal of the projects is to approximate a real-world setting as closely as possible. The projects typically run for approximately three months, with a typical workload of $\approx$65–80 hours per student. Each project follows the Unified Software Development Process [33]. The methodology is influenced by agile practices, with two-week iterations ("Sprints") and a final fixed deadline for product delivery. The students manage the projects autonomously, with a staff member assigned to each project for oversight. For issue tracking and version control, teams must use common ALM tools, such as Redmine or GitHub. Most teams use additional tools, such as Wikis, automated testing and deployment (continuous integration/delivery), or shared documents for, e.g., the requirements specification, implemented architecture, meeting minutes, or customer communication. We classify the rich set of artifacts captured during the process into type-I and -II data (see Subsection 1.1).

4.1.3. Students, researchers, customers, and the organization

All students involved studied at the second cycle (master's level). The number of students in each project is shown in Appendix B. The customer for each project came either from the university (the same or another faculty) or an external company. Since the projects were conducted in an academic context, the organization was the University of Western Bohemia in Pilsen. Instead of monetary interests, however, the university's interests are ensuring high educational standards (especially by studying the outcomes), as well as preparing the students for a potential future work environment.

Rater 1. The first rater is currently a doctoral student. Their main interest in the study is the detection of project management anti-patterns through the usage of ALM data. The first rater has about five years of industry experience working at DHL, predominantly with help and advice on using ALM tools (e.g., IBM Engineering Lifecycle Management) and consulting on software processes. They were previously involved in three process audits for industry, mining ground truth (quality of their processes or compliance to norms; using similar data plus interviews). These audits lasted for a few months, approximately one year, and one for approximately four years. The first rater worked as a mentor in the advanced software engineering course for nine consecutive years. On average, they were in charge of approximately five projects (with a range of three to eight). It is important to note that their mentor role is purely passive. They never actively attempted to avoid (or alleviate) the Fire Drill anti-pattern (or any other malpractice, for that matter). This becomes evident because some projects showed strong manifestations of the Fire Drill phenomenon (or other anti-patterns). The students could (and did) choose to ignore mentors' advice. Some students have previously failed the course.

Rater 2. The second rater holds a master's in software engineering and did not previously participate in the course as a mentor. The second rater is a former doctoral student with a main focus on project data analysis. They have over three years of industry experience working at Unicorn[6] as a project manager. Rater Two never had any active role in conducting the software engineering course.

Rater 3. The third rater also holds a master's degree in software engineering. They were previously enrolled in a doctoral program for about four years. The main focus of their third-cycle studies was the compatibility of architecture components. With more than four years of industry experience, rater three works at the Finnish software company Yoso[7], where the main customer is the Finnish state. Their role is that of an architect and a lead developer, as well as project analytics and -management (e.q., requirements analysis). They are also the head of the company's local branch in Pilsen. Rater Three never had any active role in conducting the software engineering course.

Assessor. The assessor's role was to analyze, interpret, and aggregate the raters' findings independently. They are currently enrolled in a doctoral program and hold a licentiate degree. Their main focus of study is software and information quality, with a strong focus on statistical learning and mathematical optimization. The assessor has approximately four years of prior, predominantly agile industry experience, working as a systems developer (at SAP Research[8]), lead architect, project manager, and product owner (at Softwerk[9]). As such, they have first-hand experience with agile product development. The assessor

[6]Unicorn: About Company. 2023. https://unicorn.com/en/company-profile
[7]Yoso: Company Homepage. 2023. https://www.yoso.fi/
[8]SAP: About Innovation and Research. 2023. https://www.sap.com/about/company/innovation.html
[9]Softwerk: About Company. 2023. https://softwerk.se/en/about-us

also took similar roles as a teaching assistant in two unrelated first- and second-cycle agile software development courses over the course of three years (assuming a role in between two and eight projects simultaneously in each course). The assessor never had any active role in conducting the software engineering course.

4.2. Embedded unit selection

The natural choice for the (sub-)unit of study is student projects. We initially also considered studying the Fire Drill in open source or industrial projects, but this would probably have been a purely archival study due to the lack of a real-life context [54, 57]. Furthermore, the social context of open source projects is hardly observable, and the ALM data are notoriously incomplete or often absent entirely due to the absence of (proper) project management. The latter type of project is difficult to obtain, both in quantity and quality. Since the Fire Drill is a phenomenon that affects a project in its entirety, we regard quantity as slightly more important (for this reason, we conduct an embedded case study).

The benefit of using the student projects to study our case lies in the accessibility of the associated ALM data throughout the entire lifecycle of each project. As course runners, we directly observe and record the day-to-day realities of each project. This additional data and knowledge are crucial for an accurate assessment of the ground truth. Our data spans 15 projects, four of which were conducted in 2019, five in 2020, and six in 2021, between March and June each year. We have previously detailed the various data types available for each project (see Subsection 1.1). We only included projects that provided full access to the ALM data. Therefore, a few projects had to be excluded. The non-availability of data can be primarily ascribed to the usage of external, proprietary ALM tools, sometimes demanded by industrial customers' non-disclosure requirements.

4.3. About the data

All data used in this study were previously made available as an anonymized open-access dataset [62]. It was used and made available only in digital form and was recorded exclusively digitally. All data are associated with application lifecycle management, that is, data related to application governance, development, and operations [63]. Therefore, most of the data are produced automatically by the required usage of ALM tools. The dataset includes all original artifacts for each project. To avoid introducing translation bias, textual artifacts recorded in Czech were kept pristine. The dataset also includes data collected by observation, such as mentor notes, meeting minutes, and retrospective records. The data was then digitized and added to the set.

The immediate purpose of the data collection was evaluation and grading, and the data are initially kept for no specific reason other than archival records (the university recommends keeping such course-related data for administrative purposes). The data are also kept and accessed while the projects are still running because it needs to be accessible during or after iteration reviews. Due to the absence of a specific reason, the recorded data are not specific to the Fire Drill phenomenon, as everything of potential relevance was kept or recorded. Therefore, the data can be used in a future study using a different case and/or research questions.

Since the data stem from software engineering projects conducted at a Czech university, the projects' and their data are subject to Czech legislation. As for ethical considerations, the legislation states that any student work done as a course assignment can be freely used

by the educational institution for the purposes of its main functions, research obviously being one of those.

4.4. Qualitative design

Approaching (anti-)patterns like the Fire Drill is first done through the available phenomenological descriptions. The multiplicity of evidence is first investigated qualitatively on a per-project basis. We draw on methods for data source- and observer-triangulation to obtain robust ground truth estimates [29, 64].

The primary goal of the qualitative analysis was to understand if, how, and to what degree the Fire Drill manifests in each of the projects. To achieve this, the type-I data were subjected to manual and individual inspection by each of the three raters. Before individual analysis, a common understanding was established based on the available phenomenological descriptions (see Subsection 3.1) between the raters. The raters would then go ahead and extract evidence for the presence (and absence) of a Fire Drill (called the *raters' notes* in the dataset [62]). The notes reflect which symptoms and consequences are present, how severely they manifested, and how often they were observed. They also include findings that are counter-indicative of a Fire Drill. To date, the detection of (anti-)patterns has been subject to qualitative evaluation in practice.

After a complete evaluation of a single project, a rater would then indicate an overall severity using a linear numeric rating scale of 0–10, where 0 indicates the absence of the phenomenon and 10 the strongest possible manifestation. This assessment would then serve as ground truth in the subsequent use of type-II data (e.g., variable importance, regression model, etc.). We chose to assess the severity on a project level, since a Fire Drill, according to its existing descriptions, concerns a project over its full lifecycle. Furthermore, the Fire Drill's descriptions prohibited the use of a proper ordinal rating scale, such as a descriptive, verbal rating, or Likert scale. This is because the described symptoms and consequences (see Appendix C) do not come with a severity attached. For example, the second symptom/consequence, `SC2`, reads *"only analytical or documentational artifacts for a long time"*. Therefore, in the absence of a proper ordinal scale, the raters' subjective severity assessment formed only the basis for three subsequent analyzes.

The first analysis is to measure *inter-rater agreement*. As the ground truth assessment of the raters is subjective, the only way to objectively measure the proportion to which they agree is to use some agreed-upon scale or *benchmark*. To calculate the inter-rater agreement between more than two raters, Cohen's Kappa cannot be used. Instead, for example, one of Conger's, Fleiss', or Gwet's Kappa coefficients is required. We chose to report Gwet's "AC_1" Kappa coefficient, which outperforms other coefficients in terms of having reasonably small biases for estimating the true inter-rater reliability [66]. It is especially applicable in the presence of high agreement (which, as it turns out, is the case) because of its comparatively low bias. To benchmark the computed Kappa, we apply the widely used scale of Landis and Koch [67]. However, we should note that the proposed scale by Landis and Koch was arbitrarily chosen and that each Kappa is a point estimate associated with a probability distribution and a margin of error [68]. Therefore, it is recommended to properly benchmark the raw Kappa coefficients. Gwet suggests computing the probability that a Kappa falls into a certain range by integrating a standard normal distribution (where the Kappa coefficient is the mean and its associated error is the standard deviation) [30].

The second analysis was a common session between the raters conducted to reach a consensus on their rating, using the well-established Wideband Delphi method [69, 70]. In

cases of diverging assessments, each rater presented arguments for their estimate, leading to a discussion, a repeated inspection, and a reconsideration of the information sources, until a final assessment was mutually agreed upon. Ties, rounding, and dissents were settled by the second rater since they never had any affiliation with the projects (e.g., as a mentor) and have the longest industry experience as a project manager to date. The raters were free to assign a final consensus value below or above their initial rating if there was sufficient reason to do so after the follow-up investigation of the data. The use of multiple experts is an effective measure to reduce subjective bias commonly introduced by expert-judgment software estimates [15]. The first two analyses were, in part, designed to guarantee a minimum quality of the obtained ground truth, as it is crucial for the analysis of the type-II data. Without the precautions implemented, robustness, reliability, and accuracy cannot be ensured otherwise [31].

The third analysis is a systematic approach that uses a well-defined ordinal scale to identify the prevalence of individual symptoms and consequences. Unlike the previously used numeric severity rating scale, the ordinal scale used here is of descriptive nature. Although each consecutive item represents a severity higher than that of the previous item, a linear increase is not implied. Severity is assigned or upgraded purely by description. The assessor (see Subsection 4.1.3) is to first classify each of the raters' notes and comments according to this scale. Furthermore, the assessor is to assign each observed empirical instance to one of the Fire Drill symptoms and consequences. This is done to allow us to answer the question of how the fire drill manifests itself in the projects empirically. Some of the observations, even though they are clearly related to our understanding of the phenomenon, may warrant a new superordinate symptom and consequence, especially if they cannot be assigned purely or only poorly to any of the existing symptoms and consequences.

After the first pass, a second pass is performed. In the second pass, observations (called empirical instances of a symptom or consequence and abbreviated as `ESCxx`) are conditionally aggregated and checked for data and/or observer triangulation, and the severity is adjusted accordingly. The following scale was used in both passes:

`[0]` **None:** Not at all a problem: it applies mostly to false positives (e.g., a typical symptom that was caused out of the studied context and had no adverse effects).

`[1]` **Miniscule:** Only slight indications of typical symptom(s) identified by at least one rater.

`[2]` **Minor:** Multiple indicators and/or measurable/documented symptom(s); seldomly corroborated by another rater.

`[3]` **Moderate:** Clearly identifiable and reoccurring symptoms or direct corroboration.

`[4]` **Significant:** Like moderate, but the higher severity is evident through additional data- or observer-triangulation.

`[5]` **Serious:** Agreement on the (recurrent) severe manifestation of a symptom by observer triangulation (often all raters) and/or data triangulation.

4.5. Quantitative design

The primary goal of the quantitative analysis was to facilitate the rich corpus of quantitative artifacts that are produced mostly automatically as a byproduct of conducting the projects and using the ALM tools. The goal is to enable quantitative data to contribute to the current phenomenological understanding and to automate expert-based post mortem assessment. The quantitative analyses facilitate the qualitatively won ground truth and type-II data exclusively. The available type-II data can be split into two main sources:

source code and **issue-tracking.** This split is maintained throughout all analyses because, in reality, there might be access to only one of them. The won ground truth enables a wide range of statistical analyses and supervised learning. Quantitative data in the context of (anti)patterns are rarely useful. For example, consider the number of bugs over time. For this information to be useful, we would at least need some thresholds, and those would need to be universally valid. To address our objectives, hypotheses, and research questions, we also performed three analyses (Subsections 4.6 through 4.8).

4.5.1. Activities in issue-tracking data

The Fire Drill and many related or similar phenomena are sensitive with regard to a certain balance of particular activities at any given point in time. For issue-tracking, we model three activities from the type-II data that, supposedly, are closely related to the activities described in the Fire Drill. These activities are as follows:
REQ: Activities related to requirements, analysis, and planning.
DEV: Time spent on development (implementation), testing, and bug-fixing activities.
DESC: Descoping; Effort that was planned for DEV, but never spent on it (i.e., the difference between the scope agreed on and delivered).
For each of the activities, the issue-tracking data provide timestamps (when an instance of the activity occurred) and duration. The title and description of the tickets were used to classify the issues into REQ and DEV, which were obvious choices to detect the Fire Drill. If there was only the slightest doubt, the issues were left uncategorized and not used. DEV reflects only adaptive engineering (i.e., adding features) because in these projects almost no maintenance activities are expected. Maintenance is only rarely done because there is no proper quality assurance and the delivery of agreed-upon requirements is of the greatest importance. Also, there are usually no or only a few ancillary functional requirements, such as response time, user-friendliness, or documentation artifacts. Therefore, activities other than forward engineering and bug-fixing were not considered. We also considered the frequency of the bugs, but it is unreliable because the reporting is not rigorous, often inconsistent, and sometimes completely absent. For example, many bugs were change requests in reality because the requirements were understood wrong. DESC, however, was constructed as it was deemed to be a valuable indicator for typical Fire Drill symptoms: Student projects have a hard deadline, so descoping happened frequently. The candidate solutions to a to-be-missed deadline are descoping, re-negotiation of the time frame, or overstraining people. Of these, descoping presents the *refactored* solution, while the others would exacerbate the situation. Therefore, descoping was the only allowed solution in this case.

4.5.2. Activities in source code data

From the source code, we model three types of activities from the type-II data as well. Source code is, compared to issue-tracking, a more objective source of information, because the data is not subject to human error and resulting inconsistencies (e.g., mislabeling of tickets). The source code repositories and commits thereof do not usually provide any form of annotations that would allow one to understand what kind of activity some committed work may relate to. Although it is possible to reference issues in commit messages, this feature was not used to label commits. Furthermore, we already derive three other activities from the tickets directly. By analyzing each commit's *source code density* [53], we can predict

its associated maintenance activity [52] with great confidence. The three maintenance activities are as follows:

A: Activities related to adaptive/forward engineering (adding new features).

CP: Activities related to either corrective (fixing faults) or perfective (improve or change code to accommodate future features) work.

$\mathsf{FREQ_{nm}}$: Overall commit frequency, regardless of the associated maintenance activity.

While the used classifier can differentiate between corrective and perfective commits, the Fire Drill is not described using this distinction. Therefore, we combine these two activities into a single one. Although the overall commit frequency could be designed as a weighted mixture of A and CP, we chose not to do that. Instead, the frequency is designed by disregarding all labels, similar to CP, which does not make a distinction between corrective and perfective. Therefore, the variable is called $\mathsf{FREQ_{nm}}$, where the suffix "nm" indicates a non-mixture. The association between maintenance activities and the activities as described by the Fire Drill is likely to be worse than it is for issue-tracking. However, the increased objectivity of the source code data may give an edge to these data over issue-tracking data.

4.5.3. Modeling of activities as probability densities

We have previously identified and selected activities to be modeled, for which a proper representation has to be chosen. As a single consistent representation, we model each activity as a probability density function (PDF). This is similar to how the work distribution for certain workflows in certain phases is represented in the rational unified process [34]. The density reflects the timely accumulation of activities relative to each project's lifecycle. Since we do post mortem evaluation, the time between the first and last instance across all activities can be used to normalize all occurrences' timestamps into the range $[0, 1]$ after the project end. The temporal accumulation is estimated using kernel density estimation [71]. For both sources of data, source code and issue-tracking, we know the timestamp for which an instance of activity occurred. However, for the former, there is no indication as to the duration of each instance. For the latter, however, the duration of each instance is factored in as a relative weight when estimating a density, leading to a more accurate representation of the time spent.

4.5.4. Deriving features from activities

For each project, we have previously defined what activities we model and how. However, we have not yet derived any features. Generally, a feature is a measurable property with discriminatory power, which can be used in statistical analyses and machine learning. Since temporal accumulations of activities are modeled as probability densities (in case of issue-tracking also considering the duration using weighted estimation, see Subsections 4.5.1 through 4.5.3), we choose two major types of features. The first type of feature is the amount (cumulative probability) of a certain activity that happens in a segment of a project. Its value can be determined by integrating the relevant interval of the activity's probability density. The second type of feature is the difference between any two activities on a segment that can be calculated using a (symmetric) divergence of their associated probability densities (denoted by the operator $|0$). In addition, we choose to subdivide each project into ten equally long segments. As the project time is normalized, we end up with a set of segments $\{[0.0, 0.1], \ldots, [0.9, 1.0]\}$. The original Fire Drill description only vaguely indicates

(the length of) phases. Brown et al. [11] reports an anecdote of a typical project with improper burndown for about six months, followed by a Fire Drill of four weeks. However, this is in no way representative. Therefore, we choose said subdivision. We argue that ten segments will yield sufficient precision beyond Brown et al.'s two-phase model. Furthermore, the number of segments could be arbitrarily increased to further boost the precision, if desired.

The amount θ of a certain activity ACT in a segment $[a, b]$ is the integral of its associated probability density f_{ACT} over that interval. The probability densities of the activities are designed such that $\int_0^1 f_{\mathsf{ACT}}(x)\,dx = 1$, i.e., they integrate to one, the cut-off beyond the actual project time is sharp, and the PDFs of any two activities provide *absolute continuity*. Therefore, $[\sum \theta_i] = 1$ (where $i = 1 \ldots 10$ is the segment index). The amount of a feature in a segment can be directly interpreted as a percentage (the cumulative probability of observing the related activity in the segment).

The divergence between any two activities is commutative, that is, A diverges from B the same as B diverges from A. Therefore, a *symmetric* divergence is computed. The rationale behind this is rather practical. For the modeled activities, there is no distinction between the two mutual divergences. Furthermore, if $A \,|0\, B \neq B \,|0\, A$, the result would be twice the number of features. For three activities A, B, C, the resulting set would have a cardinality of six. However, for a symmetric divergence, only $A \,|0\, B$, $A \,|0\, C$, and $B \,|0\, C$ need to be computed. We choose the Jensen–Shannon divergence [72], which is a symmetric divergence (1). It is based on the Kullback–Leibler divergence $\mathrm{KL}\,(P \,|0\, Q)$. For two continuous random variables P, Q with probability densities p, q, the divergence is computed as follows.

$$\begin{aligned}\mathrm{JSD}(P \,|0\, Q) &= \frac{1}{2}\,\mathrm{KL}\left(P \,\middle|0\, \frac{P+Q}{2}\right) + \frac{1}{2}\,\mathrm{KL}\left(Q \,\middle|0\, \frac{P+Q}{2}\right) \\ &= \int_a^b \frac{p(x)}{2}\,\log\left(\frac{2\,p(x)}{p(x)+q(x)}\right) \;+\; \frac{q(x)}{2}\,\log\left(\frac{2\,q(x)}{p(x)+q(x)}\right) dx. \qquad (1)\end{aligned}$$

The rationale behind computing segment-wise divergences is our assumption that the Fire Drill is sensitive with regard to a certain balance of particular activities at any given point in time. Therefore, we regard the divergences as an effective measurable property for observing such (im-)balances. Unlike the amount of activity, the divergence between two activities cannot be interpreted in a straightforward way and requires normalization.

4.6. First analysis: weighted mixtures

The first analysis examines the temporal accumulation of activities as is typical for when a Fire Drill is present in a project. For that, a *weighted mixture* for each activity across all projects is created. A weighted mixture is a convex combination of probability densities. In such a combination, each weight is greater than or equal to zero and the sum of all weights is equal to one (2). This is required for probability densities to ensure that no probability can become negative and the mixture integrates to 1. Recall that the raters' ground truth assessment was recorded on a numeric linear rating scale of 0–10, with 0 indicating an absence of the phenomenon. The ground truth vector $\boldsymbol{\kappa}$ can, therefore, be scaled into a weight vector by normalizing it through the division of its sum (3). A mixture for some same activity ACT across all projects (4) is then created as a weighted sum (5).

$$\sum w_i = 1 \ \wedge \ \forall\, w_i \geq 0 \ \ldots \text{ properties of a convex combination,} \tag{2}$$

$$\boldsymbol{w} = \boldsymbol{\kappa} \left[\sum \kappa_i\right]^{-1}, \text{ the normalized ground truth,} \tag{3}$$

$$\boldsymbol{f}_{\mathsf{ACT}} \ldots \text{ vector of probability densities for activity } \mathsf{ACT}, \tag{4}$$

$$g_{\mathsf{ACT}}(x) = \sum w_i\, f_i(x), \text{ weighted mixture for activity } \mathsf{ACT}. \tag{5}$$

The result of this analysis will show us, for each activity, how it typically unfolds over the lifecycle of a project that is affected by a Fire Drill. The expectation is to observe a correlation between these activities and those as described, thereby establishing an additional, symptomatic, and quantitative understanding of the phenomenon. Although we currently have only $n = 15$ projects, each project added to the mixture will lead to a more accurate representation of the activities in the presence of a Fire Drill. With a sufficient amount of projects added in the future, the weighted mixtures may become their own, quantitative pattern description of the Fire Drill.

4.7. Second analysis: variable importance

The second analysis estimates the variable importance. The term *variable* can be interchangeably used with *feature*. It does not refer to a random variable, however. The variable importance is often used as the basis for selecting features that shall be part of the training [37]. Here, we compute it for a different reason, though, that is to enrich the existing phenomenological descriptions from a quantitative point of view. The results of this analysis are not used for adaptive training. We should note that the variable importance is specific and sensitive to the model with which it was computed. For example, a neural network will estimate it differently than partial least squares. Therefore, we average the variable importance as obtained from five different models, each repeated 100 times, to get a more solid understanding. The five models used are a boosted generalized additive model [73], a neural network [74], (generalized linear) partial least squares [75], and bagged CART [76].

In Subsection 4.5.4 we have described which and how we model activities, and what kind of features were engineered. To recall, two types of features, namely the amount of an activity and the (symmetric) divergence between two activities are used. Furthermore, the subdivision into ten equally long intervals was applied. In summary, for a single activity such as REQ or CP, the amount of activity in some segment is a single feature. Therefore, for either source of data (source code or issue-tracking), we modeled three activities each and segmented them into ten intervals, having two separate features (amount and divergence) per interval ($2 \times 3 \times 10 \times 2 = 120$). Therefore, the second analysis will answer questions the first could not answer. For example, which are the most (least) important segments (or phases) of the phenomenon, or whether the balance/divergence of activities is a more suitable predictor than the number of activities. The variable importance, therefore, complements the weighted mixtures of each activity. It cannot, however, answer as to which source of data, source code or issue-tracking, is more important (that is, more apt to predict an accurate severity). That is because variable importance is estimated on either data source exclusively, as we maintain the split.

4.8. Adaptive training

The adaptive training, while also having analytical properties, is in direct correspondence with the main objective of this study: *To automate the post mortem severity assessment.* As a byproduct, we shall also learn, for example, whether and which source of data, source code or issue-tracking, is apt for use in a predictive model or what type of model works best. The design of the adaptive training is subject to the propositions that a ground truth with sufficient precision can be obtained (**Pr. 2.1**), and that there exist some artifacts among the type-II data suitable for (adaptive) training (**Pr. 3.1**). We define the *adaptive training* to be a kind of stability analysis, a process to which training data is continuously added until the chosen *stability criterion* of sufficient confidence is satisfied [32]. The two principal quantities that we will consider are the empirical generalization error or risk, R, and the confidence in obtaining predictions close to it.

4.8.1. Notations

We use notations very similar to those of Bousquet and Elisseeff [32]. Only symmetric (agnostic w.r.t. the order of the training samples) learning algorithms that produce a mapping from some input or feature space $\mathcal{X} \subset \mathbb{R}$ to some output space $\mathcal{Y} \subset \mathbb{R}$ are considered. All training is supervised. Hence, a training set S (6) consists of m tuples in $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, drawn i.i.d. from the unknown population D. An algorithm A, once fit, becomes the *hypothesis* $f : \mathcal{X} \to \mathcal{Y}$. Therefore, it is a function from $\mathcal{Z}^m$ into the hypothesis space $\mathcal{F} \subset \mathcal{Y}^{\mathcal{X}}$. We use the notation A_S to indicate that A was trained on S.

$$S = \{z_1 = (x_1, y_1), \ldots, z_m = (x_m, y_m)\}, \tag{6}$$

$$c : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+ \cup \{0\}, \tag{7}$$

$$\ell(f, z) = c(f(x), y). \tag{8}$$

The goodness of a fitted algorithm is evaluated using a cost function (7), which quantifies the difference between the true and predicted outcome. The loss of a fitted algorithm (a hypothesis f) with respect to an example $z = (x, y)$ is defined as in (8).

4.8.2. Stability analysis

The stability criterion chosen for a learning algorithm depends on the goal. In this study, the goal of the stability analysis is two-fold:

1. Select a champion model by the lowest empirical risk, among various configurations of learning algorithm, training data source, applied pre-processing, and used feature types.
2. Approximate empirical confidence intervals based on the probability that the champion model will predict an error that deviates from its expected error.

For (A), we require a model that satisfies the principal objective of predicting the severity of the Fire Drill on previously unseen projects with *sufficient confidence*. For that, we intend to repeatedly evaluate a large grid of various dimensions, in order to obtain robust estimates for the empirical generalization error for each setup (e.g., used model type, data source, etc.).

Instead of setting a threshold for what constitutes sufficient confidence, we select to tie the second goal, (B), to the amount of available ground truth once a champion was found since this is a source of actual scarcity. From pursuing the first goal, (A), the many

repetitions for each constellation resulted in a set of validation errors. This set is regarded as a separate random variable, V. Using this notion, a confidence interval can take two possible forms. The first is to express it in terms of a probability that the model estimates an error that deviates less or more from the expectation, given some bounds for the lower (d_a) and upper (d_b) deviation (9). The second is to express it in terms of a deviation that corresponds to a given lower/upper probability (p_a, p_b), that is, given a probability, a function that indicates the maximally expected deviation from the expectation in either direction (10). We can conveniently express the two notions using the probability density function (PDF), cumulative distribution function (CDF), and percent-point function (PPF) of V.

$$f_V,\ F_V,\ f_V^{-1}\ \ldots\ \text{PDF, CDF, and PPF of } V,$$

$$\mu_V = \mathsf{E}[V] = \int_{-\infty}^{\infty} x\, f_V(x)\, dx,\ \text{the expectation of } V,$$

$$g(d_a, d_b) = F_V(d_b + \mu_V) - F_V(\mu_V - d_a), \tag{9}$$

$$h(p_a, p_b) = \sup\left\{ \left| \mu_V - f_V^{-1}\left(F_V\left(\mu_V\right) - p_a\right) \right| ,\ \left| F_V\left(\mu_V\right) - f_V^{-1}\left(\mu_V + p_b\right) \right| \right\}. \tag{10}$$

When V follows a unimodal distribution, common inequalities can be applied to estimate a confidence interval. For example, if $V \sim \mathcal{N}$, the Three-Sigma rule [77] can be applied. The following four common inequalities are ordered by their bounds, from tightest to loosest: Three-sigma rule < Vysochanskij–Petunin inequality < Gauss's inequality < Chebyshev's inequality [78–80]. If any of these should be applied, one shall first evaluate the tightest inequality for which the constraints are satisfied.

4.8.3. Training flow and model selection

The design of the training flow follows recommendations to obtain robust predictive models under the constraints of small sample sizes as given by, e.g., [81–84]. Varma and Simon and especially Vabalas et al. show that standard K-fold cross-validation (CV) produces strongly biased performance estimates, particularly with small sample sizes. This problem can be evaded by using some form of nested CV, train/test split approaches, and sufficiently many repeats. Using this approach, others have previously obtained high-performing models [85]. Due to the scarcity of our data, we oversample the dataset using the well-established synthetic minority oversampling technique (SMOTE) for regression, which has been proven to significantly increase model performance [86]. For the outer resampling of the nested CV, we define an extensive search grid. The dimensions are the following (the size of each dimension is in parentheses):

- (2) types of data source: either source code or issue-tracking.
- (6) types of models, one of bagged CART, generalized linear model [87], Gradient Boosting Machine [88], neural network, Random forest [89], and Support vector machine [90].
- (3) types of features used in training: Amount, divergence, or both (see Subsection 4.5.4).
- (2) conditionally apply pre-processing in the form of principal component analysis (PCA), in order to test a lower-dimensional input space $\mathcal{X}'$.
- (49) number of training samples (using between two and 50 samples for training).
- (50) repeats using a deterministically randomized dataset.

During each of the 50 repetitions, the entire data set is shuffled. Then, a number of training samples (m) are removed without replacement, which constitutes the training dataset $S = \{z_1, \ldots, z_m\}$. Then, a constant number of 50 validation samples is selected from the rest of the dataset. These samples are completely excluded from any training and only used to estimate the validation error using the root-mean-square error (RMSE) as a loss functional ℓ (8). The pre-processing pipeline consists of three steps: Removal of (near-)zero variance predictors, conditionally reducing dimensionality using PCA, and z-standardization of the data (center and scale). The pipeline is fitted to the training data and then applied to the validation data.

The above grid has 176,400 permutations. For each, a nested inner CV is performed. The empirical risk estimator used in the nested CV is the so-called *leave-one-out* cross-validation (LOOCV) [91]. It trains on all but the i-th instance of the training data. Therefore, it estimates the stability with respect to changes in the training set. This process shall be repeated for every $i \in \{1, \ldots, m\}$ item in S, such that $S^{\setminus i}$ is the training set without that item (11). The associated R_{loo} estimator is the average over all m possible constellations of S (12). It is known as the estimator of *error stability*, which is used as a measure of the difference between true and empirical generalization error [92]. Since the training here uses all but one data point during LOOCV training, it should be noted that the empirical estimate of the generalization error has a slightly pessimistic bias [81].

$$S^{\setminus i} = \{z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_m\}, \tag{11}$$

$$R_{loo}(A, S) = \frac{1}{m} \sum_{i=1}^{m} \ell\left(A_{S^{\setminus i}}, z_i\right). \tag{12}$$

The inner CV is repeated for a fixed number of 25 folds during which a nested CV is performed for each. Each m-th estimate during LOOCV in addition is found by conducting a nested grid search for optimal model-specific hyperparameters, which typically span between $1e^1$ and $1e^3$ permutations. For the models *gbm* and *nnet*, fine-tuned grids and fewer inner repeats are used. The many-times repeated training allows us to approximate the probability density of V, its expectation $\mathsf{E}[V]$, and its associated confidence intervals.

5. Analysis and results

In this section, we report the results of the qualitative and quantitative analyses. Those results, their validity, and limitations are then further discussed in Section 6. The reported results represent our main findings and are those primarily relevant to the posed research questions. The results here are presented in the same order as the relevant methodology: First, we demonstrate results related to the qualitative evaluation. Then, results obtained quantitatively are shown from Subsection 5.4 and onwards. The interested reader is directed to an extensive technical report with numerous additional results that extend far beyond the scope of this study [78].

5.1. Inter-rater reliability and consensus

Prior to conducting a session for finding a consensus, we assessed the inter-rater agreement. This was done by first computing Gwet's *agreement coefficients* AC_1/AC_2 [66] and then

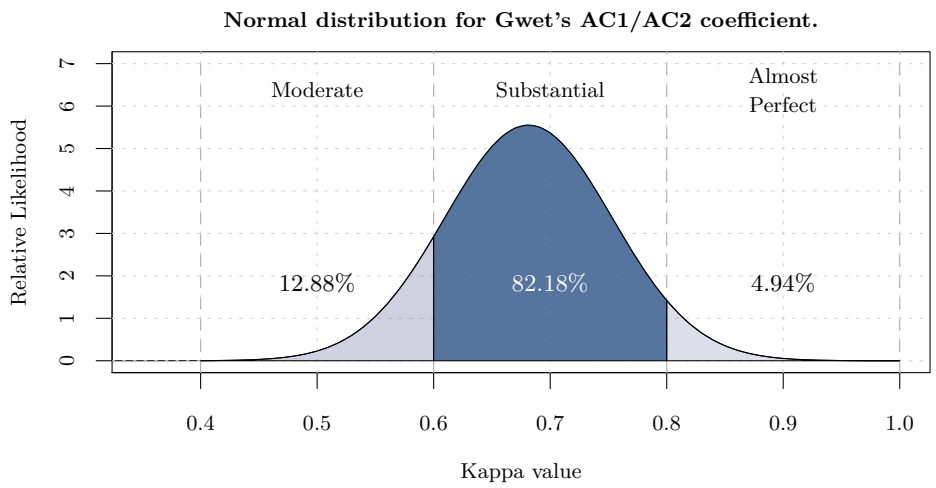


Figure 3. Gwet's AC_1/AC_2 coefficients [66] benchmarked against the Kappa scale of Landis and Koch [67]. The opacity indicates the probability for the agreement to be in the given range. The distribution's mean is at ≈ 0.681 and the standard deviation is ≈ 0.072

benchmarking it against the scale as suggested by Landis and Koch [67]. Gwet's coefficient results in a percent agreement of $\approx 86.2\%$, a by-chance percent agreement of $\approx 56.8\%$ (which it corrects for), an agreement coefficient (*Kappa*) of ≈ 0.681, a standard error (deviation) of ≈ 0.072, and a p-value of $\approx 2e^{-7}$. The p-value indicates, under any common significance level, that there is no practical corroboration for the null hypothesis of the inter-rater agreement test that the raters' agreement happened purely by chance. Instead, we accept its alternative hypothesis that chance did **not** cause the observed agreement [93]. According to the computed benchmark, the agreement is, for the largest portion of $\approx 82.18\%$, "substantial" [67]. Another $\approx 12.88\%$ of the agreement is "moderate", and some of it ($\approx 4.94\%$) is even regarded as "almost perfect." This is visualized in Figure 3.

Assessing inter-rater agreement addresses **RQ 1.1** directly. The computed benchmark indicates that most of the agreement is substantial. However, in order to solidify our answer, we also compute the benchmarks of Cicchetti and Sparrow [94], Fleiss [95], and Regier et al. [96]. Note that the latter uses the same (positive) levels as the benchmark by Landis and Koch, but only changes the labels (from "Moderate" to "Fair", from "Substantial" to "Very Good", and from "Almost Perfect" to "Excellent"). For Cicchetti and Sparrow's benchmark, the agreement is "Fair" ($[0.4, 0.6] \approx 12.88\%$), "Good" ($[0.6, 0.75] \approx 70.14\%$), and "Excellent" ($[0.75, 1.0] \approx 16.98\%$). For Fleiss's benchmark, the agreement is "Intermediate to Good" ($[0.4, 0.75] \approx 83.02\%$), as well as "Excellent" ($[0.75, 1] \approx 16.98\%$).

Finding a consensus is different from only applying, say, a weighted average. While there was close or full agreement in many cases between the raters, some cases warranted for a common, retrospective inspection of the projects' artifacts, which led to ratings that were sometimes outside the range of the individual assessment. Most often, however, the raters were able to settle on one of the proposed ratings or a neighboring value.

5.2. Phenomenon prevalence and manifestation

Here, we report the summarized results of how the Fire Drill manifests across the projects, in terms of concretely observed empirical instances of certain symptoms and consequences.

Those results address **RQ 1.2** and were obtained by systematically analyzing the raters' notes. The most recent phenomenological descriptions of the Fire Drill define seven supercategories for symptoms and consequences (see Appendix A). We refer to them using the notions `SC1` through `SC7`. The qualitative evaluation led to us defining three additional supercategories. Since these were derived from empirical observations, we refer to them as `ESC1`, `ESC2`, and `ESC3` (see Appendix C). An observation was only logged if the rater's notes allowed it. For example, some notes indicate a problem, without a cause: *"descope in later stages of the project,"* or *"poor testing"*. Here, we cannot assign an instance of a symptom/consequence. Also, if the severity cannot be decided between two levels, then the rater's Fire Drill severity is applied to reach a decision.

We should note that, due to the nature of the definition of `SC7`, no instances of it were found. Its description would affect a project rather globally. However, some of `SC[1-6]`, as well as `ESC[1-3]` convey the portrayed problematic of `SC7` in parts, so that findings were assigned to these instead. During the evaluation of the raters' notes, many instances emerged that could have been assigned to the original symptoms and consequences `SC[1-7]`. However, it became apparent that a new supercategory would perhaps be more suitable for many instances. Therefore, we introduced the notion of `ESC[1-3]`. Those reflect poor communication, high project risk, and poor usage of project management tools and methodologies, respectively. `ESC[1-3]` are not supposed to become part of the Fire Drill description. Rather, they were introduced for a more fine-grained assignment of observations to superordinated symptoms and consequences. Therefore, any observation assigned to `ESC[1-3]` could as well be assigned to the original symptoms and consequences.

Qualitative evaluation. Our findings indicate that the most reoccurring problem is high project risk (`ESC2`). It is closely followed by poor communication (`ESC1`) and a compromised project schedule or scope (`SC6`). The most infrequently observed problems (apart from `SC7`) are a spike of development efforts towards project end (`SC3`), the absence of sufficient quality assurance and project tracking during development (`SC4`), as well as the delivery of only analytical/documentational artifacts over a (too) long period (`SC2`).

Project risk (`ESC2`) is the most diverse supercategory, that is, it has the most different types of empirical observations by far (11). Risks are sometimes highly connected and emerge consecutively, such that the consequence of one problem is the symptom of the next problem. For example, an imbalance of activities (e.g., time spent on development when it had been required on requirements analysis instead) often leads to descoping, which itself caused frequent project schedule adaptations or quality regressions. Many problems can be attributed to the lack of (professional) experience in students, such as the misestimation of work items, technical difficulties (e.g., development environment, infrastructure, etc.), improper self-organization, or the (unwitting) misinterpretation of business requirements. However, not all problems were caused by students. For example, the customer representative was sometimes not available at the required capacity, thwarting the team. In other cases, the customer did not understand the technical challenges involved, which led to, e.g., unrealistic expectations or greatly diverging effort estimations.

While observations of the second-largest problem, poor communication (`ESC1`), could be attributed in many cases to project risk, many problems would go underappreciated if doing so. The most frequent observation was an unresponsive customer or unsatisfactory communication. It would often manifest through delayed, slow, or incomplete responses. Sometimes, critical material, decisions, or information were imparted too late, causing other problems, such as a stalled team or compromised schedule. This observation is perhaps the most prominent for one of the root causes of a Fire Drill: management stalls development.

Poor communication can be mutual, that is, between parties, or be caused by just one party. Most of the problems can be attributed to the mutual category. However, there are also instances of students not renegotiating misunderstandings and of customers interfering with the students' project management without telling them.

The third-largest problem, a compromised project schedule or scope (`SC6`), had three different manifestations that were observed multiple times, both in the same project as well as across projects. Most often the schedule was compromised because the students accepted change requests, the re-prioritization of existing issues, or the addition of new issues in the middle of an iteration. This situation was often exacerbated by an improper change management process. The second-most frequent observation for this problem was that work was not completed as planned and the dragging of issues into the next iteration. The reasons for these observations were multifarious, such as an over-challenged team, misestimation, or unequal work distribution. Related to this observation, but still distinct from it, is that the team gets used to and regularly accepts doing overtime (and prolongs an iteration) or truncates scope (planned work).

As for problems of the remaining symptoms and consequences, the most frequent observations were related to a slow project startup (`SC1`), especially with regard to non-developmental activities. This was most often related to a familiarization process needed by the new team, such as familiarization with the other members, new or changed tools, ways of communication, or yet-to-be-improved early-stage procedures. Another common cause was an unclear scope, such that the team went into procrastination (underscoped) or did have a hard time finding their way into the project (overscoped). `ESC3`, the poor usage of project management tools and related methodologies, constituted the next bigger class of symptoms and consequences. Typical problems are attributed to the improper usage of ALM tools, such as using wrong item types (e.g., marking an Epic as a Task), not breaking down large deliverables and features, or careless and imprecise logging leading to a discrepancy between logged and done work. The teams were allowed to use additional tools for information and knowledge management, which led to confusion and unnecessary duplication. Lastly, `SC5` concerns the final product, its quality, and delivery date. The quality was compromised in some cases by skipping over planned features or proper quality assurance. In some cases, the product was completed but delivered only after the final due date.

The refactored solution to a Fire Drill includes measures applicable for when there is time (iterations) and resources left. While some projects showed increasingly stronger signs of the phenomenon, no intermittent measures were implemented to alleviate the problems. It was only towards the very end that a solution had to be found. This is attributed to our context: While there are deliverables after each iteration, the customer is primarily interested in the final product, as feature-complete as possible. The students are primarily interested in passing the course, concentrating their focus on the final delivery, too. Refactored solutions to ameliorate an eventuated late-stage Fire Drill are to re-negotiate the delivery date, to triage the remaining budget into implementing missing features and quality assurance, or to descope. Due to the lack of monetary interests (and related pressure) on the customer's side and the impossibility of changing the delivery date (constraint of the context), the Fire Drill manifested predominantly through descoping, which proved to be a valuable predictor later.

Quantitative evaluation. We report on the average severity for each symptom and consequence, as well as on the total severity. The utility of the average severity is a ranking of the supercategories, that is, to determine what the common severity of the manifestation for each superordinate category is. The total severity is the sum of the observations' severity.

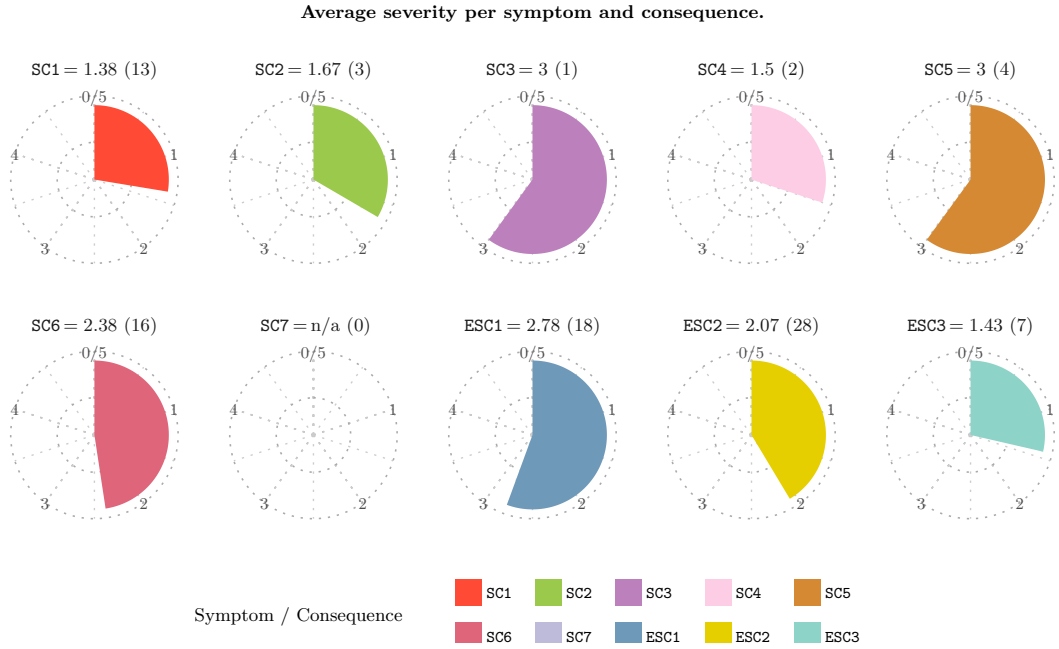


Figure 4. Average severity for each (empirical) Symptom/Consequence.
Also shown is the number of observed instances in parentheses

Its utility is more specific for learning for our concrete case, as it should give an indication as to which symptom and consequence is the most pronounced in our projects.

Figure 4 shows the average severity per symptom and consequence. `SC[2-5]` have only a single or few observations, which should be considered when ordering supercategories. `SC1` and `SC6` make for a good comparison, for example. While both have similarly many observations, the latter has a significantly stronger average severity. Encountering instances of either symptom and consequence might be similarly likely, but an observation of the latter indicates a worse case of a Fire Drill. The strongest average severity is exhibited by `ESC1` (disregarding `SC3` and `SC5` which have only a few observations) with a value of ≈ 2.78. Recall

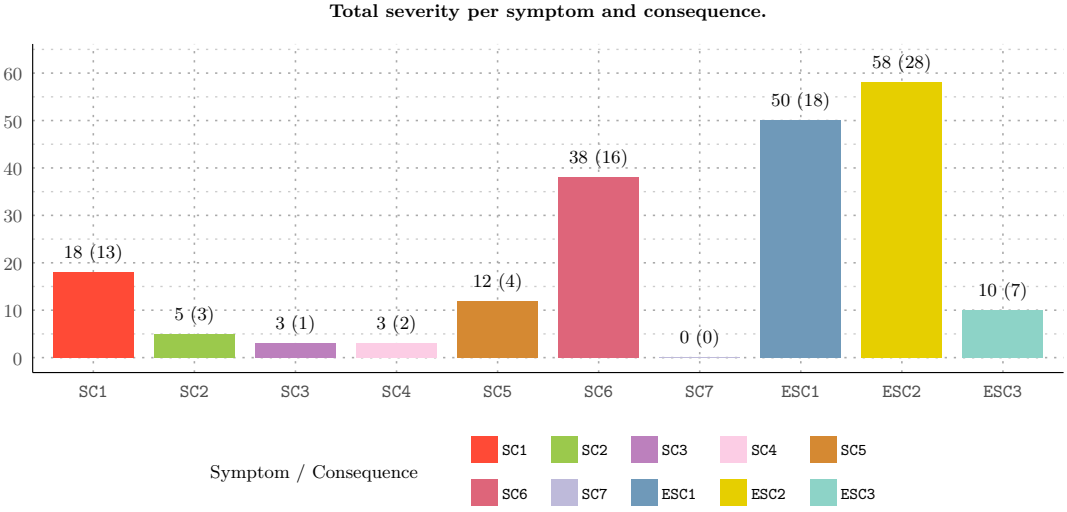


Figure 5. Total severity for each (empirical) symptom/consequence.
Also shown is the number of observed instances in parentheses

that a value of 3 corresponds to the ordinal level "Moderate" (see Subsection 4.4), which is characterized as clearly identifiable and reoccurring symptoms or direct corroboration. Since its average severity exceeds that of SC6, the poor communication between stakeholders comparatively appears to be the worst of all the problems in our case.

This impression slightly shifts towards ESC2 when looking at the total observed severity (see Figure 5). Ten more instances (28 total) were observed of ESC2, which explains its high total severity. In other words, while poor communication is comparatively worse when it happens, the projects were substantially more often subject to some form of risk. A third, quite evident problem is a compromised schedule or scope (SC6). By a larger margin, this is followed by problems related to a longer stall during the project beginning (SC1), a final product with low quality (SC5), and poor usage of ALM tools (ESC3). The low count of observations for SC[2-5] also results in a comparatively low total severity.

5.3. Phenomenon absence

The evaluation of the raters' notes also exhibited evidence for the absence of a Fire Drill for a number of projects. In Appendix D, we have gathered a list of observed symptoms and consequences in order to answer **RQ 1.3**. This list is comprised of symptoms and consequences that are *counter-indicative* to what constitutes a Fire Drill. While the absence of evidence is not evidence of absence, the gathered items should be more regarded as *true negatives*. This list does not claim to be complete, nor is the presence of a single item (or few) sufficient proof for the phenomenon's absence. On the contrary, evaluation of the raters' notes indicates that a project may exhibit symptoms and consequences for and against a Fire Drill, even simultaneously. For example, communication and collaboration with the customer might be seamless (as indicated by the counter-indicative symptom and consequence CISC01), but the project's schedule might still get compromised due to misestimation caused by inexperience. Sometimes, there are also signs for the opposite of a Fire Drill (or other patterns). Our main observation here is essentially related to an underchallenged or underutilized team, and often related to the quick completion of work items, delivery before the deadline, a too-simple product, or skipping over quality assurance or planning and/or analysis with the direct start of implementation.

5.4. Quantitative phenomenon manifestation

The previously won ground truth allows leveraging the quantitative type-II data. With each project having a severity attached, we can find typical accumulations of (maintenance) activities that are characteristic of a Fire Drill (**RQ 2.1**). Figure 6 shows, for each activity as it is found in source code and issue-tracking data, a weighted mixture (convex combination (2) using the ground truth as weight). Three out of 15 projects had a ground truth consensus of 0 and, therefore, do not contribute to any of the weighted mixtures. Another effect of this circumstance is that the weighted mixtures quite obviously mirror the activities from a few, severely affected projects. Since each activity's mixture is an ordinary probability distribution (with integral $= 1$), we can compare them in a straightforward manner.

Source code. The activities in the source code are derived from maintenance activities (adaptive $\sim$ A, corrective+perfective $\sim$ CP) [52]. The overall commit frequency, regardless of the associated activity, is depicted as FREQ. Adaptive activities show a somewhat slow start, followed by a peak in the third quarter of the project, and a sharp decline afterward.

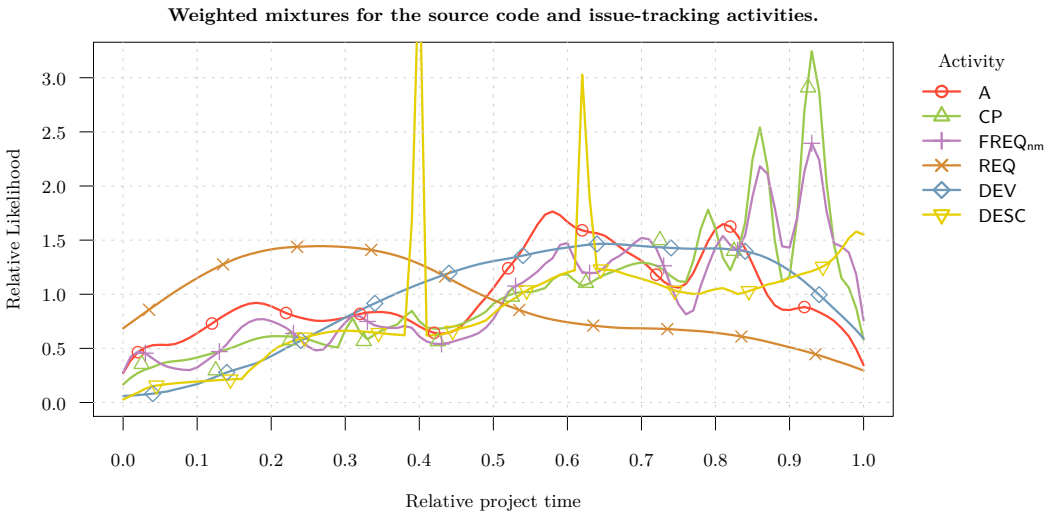


Figure 6. Weighted mixtures for each of the source code and issue-tracking activities (using the ground truth to create convex combinations)

To some degree, A follows the expectations according to the Fire Drill's phenomenological descriptions. CP and FREQ on the other hand, steadily increase for the first ≈80% of the project, only to peak shortly after at about ≈90% of the project time. The combined behavior of A and CP is counter-intuitive to the expected behavior of these activities, as per the Fire Drill's phenomenological descriptions (i.e., a significant increase of adaptive activities with a simultaneous decrease in corrective+perfective activities). FREQ, on the other hand, distinctively shows the expected peak of a Fire Drill towards the project end.

Issue-tracking. The issue-tracking activities provide us with an additional quantitative perspective. Due to the nature of the projects, DEV is expected to only reflect adaptive engineering, similar to source code's A. The qualitative evaluation confirms this expectation, because corrective and perfective activities were rarely, if at all, logged in the project management tools. Indeed, we observe a quite similar temporal accumulation between A and DEV, with a peak in the third quarter as well. More or less inversely proportional to DEV is the accumulation of activities as captured by REQ. Both of these activities are in accordance with the existing phenomenological descriptions, such as an imbalance where in the beginning of a project activities connected to requirements, analysis, and planning prevail, while development is thwarted for one or the other reason. Approximately uniform distributions for these two activities would be asymptomatic for a Fire Drill in the ideal case. The descoping activity DESC was designed by us based on the assumptions that it would make for a good predictor of the Fire Drill. We observe that this activity steadily – in fact almost linearly – accumulates from project start to end. This indicates that affected projects are subject to descoping from the beginning and that these projects do not manage to remedy this situation.

5.5. Variable importance

Another question we sought to answer quantitatively using type-II data was about the importance of activities and project phases for accurately predicting the phenomenon severity (**RQ 2.2**). For that, we subdivided the normalized relative project time into ten

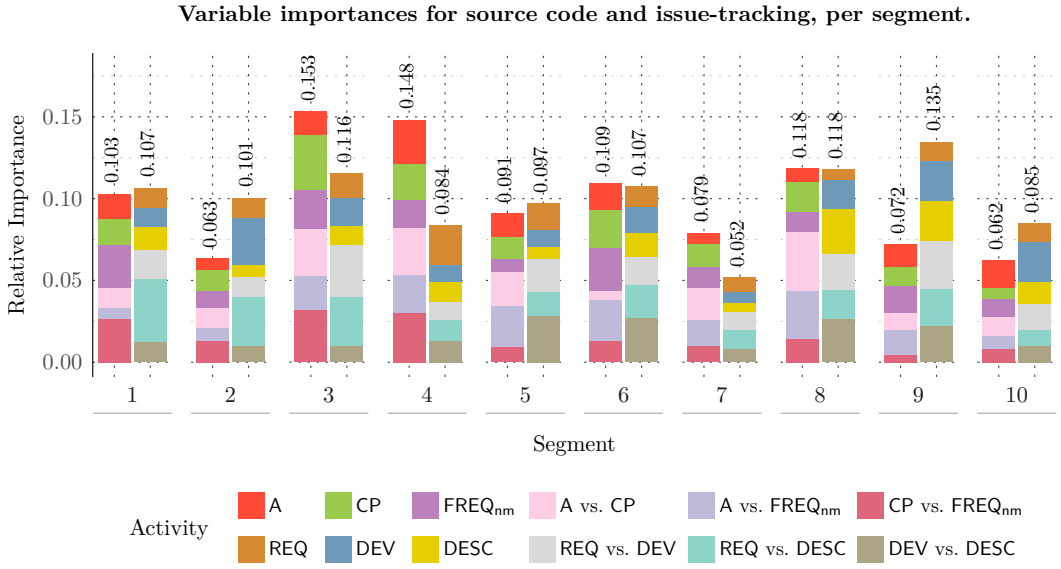


Figure 7. Per-segment and per-feature variable importances as averaged across five models, each repeated 100 times: Boosted Generalized Additive Model, Neural network, (Generalized Linear) Partial Least Squares, and Bagged CART

equally long, consecutive segments (see Subsection 4.5.3). Figure 7 shows a graphical result of this. The exact numeric results as shown in the figure are included in Tables E1 and E2, which are to be found in Appendix E. Looking at the final numbers, we can say that the divergence features have greater variable importance than the amount features, both for source code ($\approx 52.5\%$) and even more so for issue-tracking ($\approx 56.2\%$). The two most important features for either data source are $\mathsf{A}\,|0\,\mathsf{CP}$ (19%) and $\mathsf{REQ}\,|0\,\mathsf{DESC}$ ($\approx 20.7\%$). The two least important ones are `A` ($\approx 14.2\%$) and `REQ` ($\approx 13\%$).

As for the per-segment importance, the results diverge between data sources, with a larger variance for source code. There, it ranges from $\approx 6.2\%$ to $\approx 15.3\%$, with a standard deviation of $\approx 3.3\%$. The first five segments, that is, the first half of the project, account for $\approx 55.9\%$ of all importance, where segments three and four seem to be particularly important. Interestingly, segments seven, nine, and ten exhibit a comparatively low variable importance for predicting phenomenon severity. The range for issue-tracking data is from $\approx 5.2\%$ to $\approx 13.5\%$, with a lower standard deviation of $\approx 2.3\%$. Both project halves are equally important when using issue-tracking data. Except for segments seven and nine, each segment is almost equally important.

5.6. Adaptive training

The third set of results is related to the third set of propositions and hypotheses and addresses the research questions therein. According to the previously described methodology (see Subsection 4.8), we conduct the adaptive training, using a large number of constellations and repeats in order to obtain robust estimates. We find that models trained using source code data converge faster and result in a lower final training error (answer to **RQ 3.1**), that not reducing dimensionality using PCA works better for a slight majority of cases, and that using both kinds of features simultaneously leads to robust convergence. Figure 8 shows the distribution of validation errors for all six models when trained on 20 instances. When we

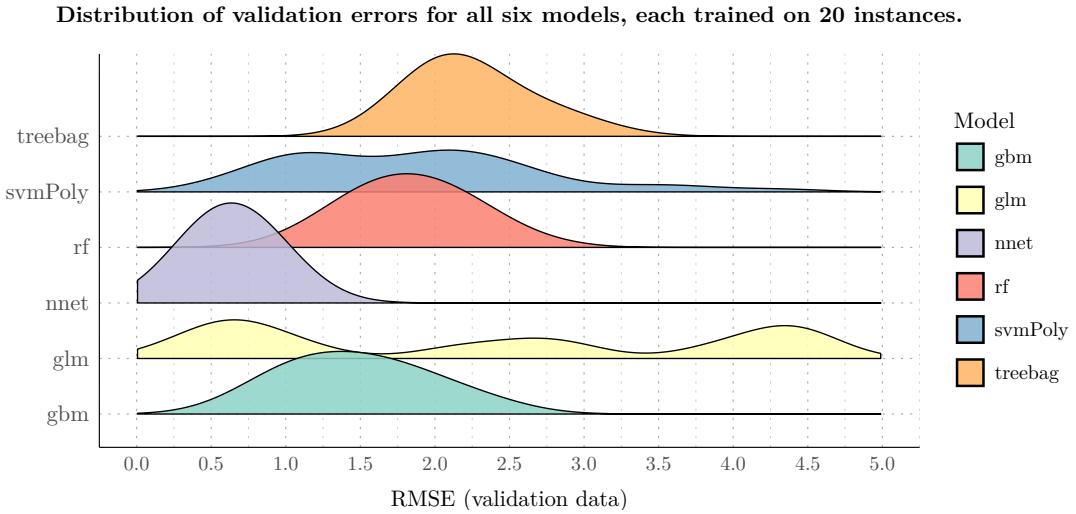


Figure 8. Distribution of validation errors (50 instances) computed using the root-mean-square error for all six models, where each was repeatedly trained on 20 instances of the source code dataset, without applying PCA

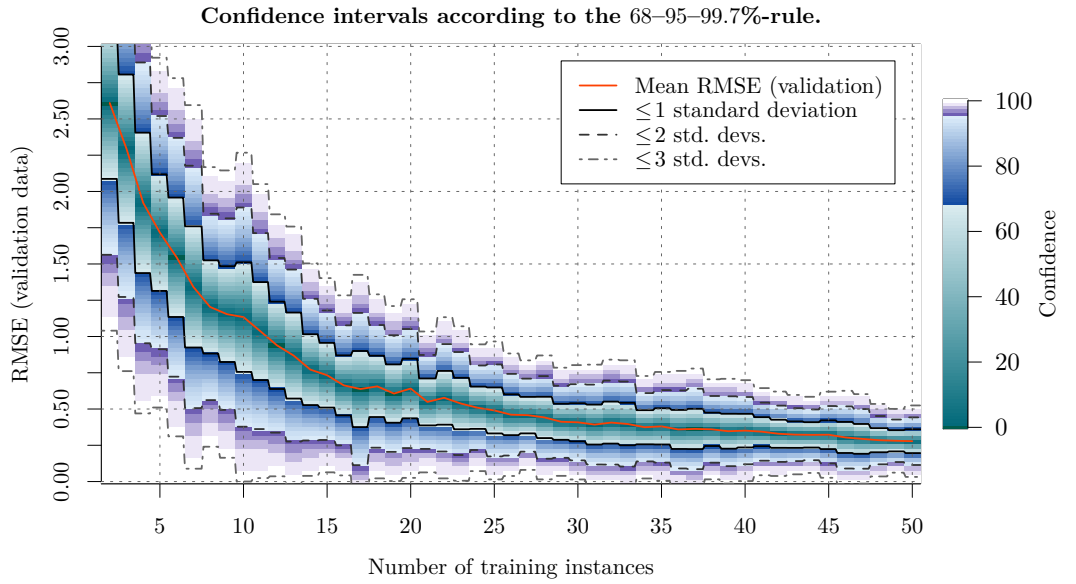


Figure 9. Continuous confidence of the neural network predictor, with regard to the number of training instances. Shown are the values according to the 68–95–99.7%-rule (assuming a normal distribution for every generalization error).
The mean RMSE was determined using 50 models' predictions on validation data.
The three color gradients correspond to the three sigmas

compare the data sources, models trained on source code achieve a lower mean (0.75/0.92), median (0.52/0.71), minimum (0.152/0.158), and standard deviation (0.58/0.65) for the validation RMSE error, summarized across all the different constellations.

The champion model is a standard feed-forward neural network that uses divergence features from source code only. This is also reflected in Figure 8, however, here it is only

shown for 20 instances. The full adaptive training of the champion model is shown in Figure 9. The many observed validation errors obtained through the numerous repeats are almost always normally distributed, indicating the robustness of the final model. That also allowed us to apply the Three Sigma rule (which has the tightest bounds) to determine a confidence interval per number of training instances and to find some answers to **RQ 3.2**. Most notably, training the champion model on 25 instances produces an expected validation error of ≈0.46, and this error will not be larger than ≈0.96 by a probability of 99.7%[10]. In other words, such a model will predict the Fire Drill severity on a scale of 0–10, and by doing so it will almost surely not deviate by more than one point on this scale (the expected deviation is even less than 0.5). The expected RMSE falls below 1.0 for the first time when trained on twelve instances. For 15 instances, the probability that the prediction will be off by maximally 1 is already at ≥76%, as can be seen from the figure. For 20 instances, this probability increases towards ≥92%. For 50 instances, the expected error plus deviation is 0.52 or less, by a probability of 99.7%.

6. Discussion

In this section, we first summarize all results and relate them to the previously established propositions and hypotheses. We then discuss the validity of our study and its results, its limitations, and address replicability and generalizability.

6.1. Summary of the results

We have shown that experts can independently agree on a subjectively chosen rating and that their agreement was substantial and did not occur by chance. This was the basis for the propositions **Pr. 1.1** and **Pr. 1.2**. The existing phenomenological descriptions of Fire Drill are sufficiently precise, and its severity can be accurately determined. It also confirmed the hypothesis **Hyp. 1.1** that the mostly unstructured type-I data provide sufficient quality for the assessment task. The agreement on the absence was also reliable (**Pr. 1.3** and **Hyp. 1.2**) so we were able to gather a list of counter-indicative symptoms and consequences. Most notably, however, the qualitative assessment and the following quantitative evaluation of the Fire Drill's manifestation allowed us to identify the most pronounced and prevalent symptoms and consequences in the considered projects. We find prime examples of reoccurring Fire Drill elements across our projects, such as instances of (chains of) project risk, poor stakeholder communication, and regressions in the form of descoping and compromised schedules. The circumstances and constraints of the studied context allow us to deduce explanations for the concrete manifestation.

For the first quantitative analysis, we compose a weighted average of how activities typical for a Fire Drill-affected project accumulate. It is deemed an appropriate representation since we previously confirmed hypothesis **Hyp. 2.1**, that is, activities will display a behavior characteristic of or conforming to the Fire Drill's phenomenological descriptions. These allow us to establish an additional *quantitative* understanding of the phenomenon. We take advantage of the wide range of digital artifacts and reliable ground truth to do so (**Pr. 2.1**). The weighted mixtures unveiled some unexpected results, which are, however,

[10]While these and the following results can be seen in Figure 9, they are calculated exactly using the previously introduced notions for probabilities (9) and deviations (10) of confidence intervals, respectively.

explained by the confines of the studied context and case and corroborated by findings from our qualitative analysis. Late-stage Fire Drills were most often addressed by descoping, which is a valid – here the only valid – *refactored* solution. This is reflected quite well in the activities found in the source code. Instead of rushing to add the last missing features, the focus is predominantly on corrective activities and then on perfective activities. We know this from decomposing CP into its separate activities. While the Fire Drill description to date does not differentiate between corrective (C) and perfective (P), using source code allowed us to gain this insight. This is surprising since the delivery of agreed-upon requirements is deliberately prioritized over maintenance in these projects. An explanation for this might be that it became apparent in some of these projects that some form of compromise of scope is inevitable, so the efforts were likely re-focused on ensuring that a minimum quality corresponding to the requirements, for the to-date existing features, was met, or that the product works at all.

The second analysis of variable importance unveiled two principal findings. First, knowing about the (im-)balance of two activities is a more important predictor for Fire Drill severity than knowing how much time was spent on the activities. This is an important finding, as the balance between two activities is a relative metric. The phenomenological descriptions, as well as the results of our qualitative analysis, both support this. This result means that, at least on average, it is of greater relevance to know how diverging any two activities are, rather than their concrete amount (cumulative probability). Rather than waiting until the project end to calculate the amount of each activity in each segment, we can observe how activities diverge in a single segment intermittently and use this as an indicator in the future. The second important finding is that segments (project phases) are, except for a few cases, almost equally important (also, no phase has (near-)zero importance, **Hyp. 2.2**).

The third analysis was designed to determine whether a robust, reliable, and *low-risk* predictive model can be obtained. The designed adaptive training workflow showed that we can obtain a suitable model from those data (**Pr. 3.1**, **Hyp. 3.1**). With sufficient confidence as our stability criterion, we demonstrate that models trained on as few as 15 instances will completely automate severity assessment in the future, within acceptable confidence intervals (**Pr. 3.2**). Both sources of data individually, source code or issue-tracking, are suitable for this task (**Hyp. 3.2**). Generally, we observe that the adaptive training flow converges nicely with increasing amounts of training data. Clearly, the stability of the model is closely related to the amount of available training data. As for **RQ 3.2**, the precise threshold for *stable* depends on the application, but we figure that to predict the severity of the Fire Drill in our case and context, a model trained on 15 to 25 instances would deliver sufficient stability, as the results would not be misleading, even when off slightly.

6.2. Validity, limitations, replicability, and generalizability

The choice to conduct an embedded case study was a natural one, given the principal objective and the previous pilot study [51]. Our study is of longitudinal character, as the same course was studied repeatedly and projects were selected over a duration of three years. Each project is unique with regard to individuals, groups, social structure, software, etc., and it is unlikely that the same set of events unfolds again in the same way [97]. Therefore, to maximize reliability and minimize bias, projects were selected from each year. Instead of conducting a case study for each project, we chose a common case and context that allows us to collect and analyze quantitative data as well, especially since we

intended to backpropagate the gained knowledge to the phenomenological descriptions and, therefore, the studied case.

A case study should be chosen when conducting an empirical investigation of a contemporary phenomenon in its real-life context, especially when the phenomenon cannot be clearly separated from its context [55]. Furthermore, a variety of data sources is required (see Subsection 1.1). The Fire Drill is a phenomenon that is *always* embedded in some (social) context it cannot be separated from, and it is not the goal of this study to propose or attempt a separation. Instead, we have made extensive efforts to minimize the impact our study may have. In order to maximize the degree to which this study can be (partially) replicated, we outline an extensive protocol (see Section 4) and publish all original data [62] and experimental designs [78]. Construct validity is achieved by a variety of measures, such as the usage of multiple data sources and observers (triangulation), ascertaining of inter-rater reliability, and controlled experiments (e.g., many repeats, model averaging, training with stability criterion, etc.). Independent raters and an assessor use systematic protocols, which allowed us to suggest and confirm chains of causality and ensure repeatability and replicability. The usage of two completely independent data sources (source code and issue-tracking) provided a second perspective that we exploited to corroborate our findings.

The limitations of the obtained results lie in the external validity and generalizability. Results such as the predictive model, weighted mixtures, or variable importance do not have validity outside the studied context, as it introduced constraints that force the phenomenon to take certain turns. We have observed a great number of concrete instances of symptoms and consequences. Although extensive, these observations cannot be exhaustive, nor can their distribution be representative outside our context. This is similarly true for the asymptomatic, counter-indicative observations. The Fire Drill shares similarities and indicators with other kindred phenomena, which are also based on temporal accumulations of activities (see Subsection 3.1.2). Therefore, we expect the external validity of the protocol suggested for studies of these phenomena. The generalizability of this study comes from subsequent and replicating case studies. For example, one study may examine the pattern "Half Done is Enough" in the same context and another one the Fire Drill in an industrial context. Only with a sufficient amount of case studies will we be able to reach a definitive understanding of the Fire Drill and how it does (not) manifest in certain contexts.

7. Conclusions and future work

We have shown that activity-based detection of complex phenomena is viable and can be used to reduce the otherwise required amount of expert-based, qualitative, and extensive analyses. Our work has several practical implications. First, instances of maintenance activities are plentifully found in typical software projects and make for highly cost-effective and robust features. Second, an existing ground truth can be leveraged to make sense out of these quantitative features. Third, predictive models using these features require only a few observations (projects) to produce low-risk predictions. Last, A well-trained model can produce predictions that are accurate enough such that they can support or (partially) replace the expensive, error-prone, and expert-based evaluations. The practical implication is that such a model can be reused on only the quantitative data (e.g., we have shown that the commits of the repository are sufficient) of future projects for predicting the severity of complex phenomena. This may be useful in circumstances where a fast

and computationally cheap analysis is required to, e.g., quickly filter and flag (for a full follow-up in-depth qualitative analysis) projects affected by a problem.

7.1. Synthesis

For an embedded case study, it is important to synthesize all results and to backpropagate them to the studied case [25]. We have presented results from each project's individual, qualitative, and idiographic study, as well as results from the quantitative, nomothetic analysis across all projects (see Section 5).

We studied the Fire Drill as is embedded within a software engineering course. For this specific case and its surrounding context, we predominantly find results that are in agreement with the phenomenological descriptions. The in-depth individual study of each project, as well as a common evaluation across projects, allowed us to find explanations for all the results diverging from it. In affected projects, we observe typical peculiarities of a Fire Drill, such as management that stalls development, or late-stage rushes. While the anti-pattern suggests renegotiating the final deadline as one solution, the students truncated the scope of their applications to mitigate the fallout, thereby implementing the only valid refactored solution within the course. We gather the most significant symptoms and consequences of a Fire Drill within the course. Evaluation of total and average severity showed that project risk, poor communication, and a compromised project schedule or -scope are the biggest problems that students encounter. We learn that a Fire Drill may affect a project as a whole, but we also observe micro instances affecting single iterations. It was previously suggested and now confirmed by us that the Fire Drill is an anti-pattern that can be the result of, encompass, or cause other, often conceptually smaller anti-patterns such as "Analysis Paralysis" or "Cart Before The Horse". We find evidence that is counter-indicative of a Fire Drill and observe projects that exhibit evidence both for and against its presence, simultaneously. Raters are able to identify this and other circumstances by being provably able to confidently agree on a severity. Our conjecture that descoping makes for a strong predictor is confirmed; projects are affected by it over the course of their entire lifecycle and the amount of time wasted on it consistently increases. We conclude that the Fire Drill is an anti-pattern that was deliberately described vaguely, but that it is still possible to derive concrete and specific problem instances from it. Observing only a few instances proved sufficient for building low-risk predictive models that can exploit activities that are modeled after either source code or issue-tracking data, as both kinds of data sources are eligible for the task.

We suggest that other phenomena that are characterized by activities that can be captured and analyzed similarly can be subjected to the methodology presented in this work. Therefore, we intend for the methodology to be the main contribution. Our empirical observations are likely valid in other similar cases and contexts, too, but subsequent (partially) replicating case studies will have to show this. Most patterns today are only described from anecdotes or other literature (phenomenological descriptions), but rarely come with a set of empirical observations attached. We contribute directly to the existing understanding of the Fire Drill by unearthing concrete empirical instances associated with its ascribed symptoms and consequences (the supercategories). More significantly, though, we have shown how to use a (scarce) ground truth to establish an additional, quantitative understanding by leveraging the available data (which was not possible previously), making the Fire Drill perhaps the first pattern-like phenomenon that is described from both perspectives. Lastly, having uncovered the most frequent and prevalent

issues that participants of the course encounter, we will attempt to incorporate the won insights into future editions of the course to improve all participants' experiences.

7.2. Future work

This study is the first to properly replicate the significant findings from the previous pilot study [51]. We intend to replicate this study with other related and similar phenomena. Candidates are, for example, Cart Before the Horse, Half Done is Enough, the Net-negative Producing Programmer, the Lone Wolf, Nine Pregnant Women (a variant of Brook's law), or Analysis Paralysis. We also encourage conducting (partial) replication studies with the same or a different phenomenon and alterations to the context, especially in industrial settings. It might also be worthwhile to consider additional artifacts found in the application lifecycle management data for data triangulation. Additional analyses, such as an earned value analysis, might provide additional corroboration, especially when its result can be correlated with the maintenance activities. Subsequent studies that analyze the Fire Drill will contribute towards a more complete picture of the phenomenon. Studies using other phenomena will also help to increase the margin between phenomena, making them more distinguishable from each other. Once a sufficient number of case studies were conducted, a multiple case study should be performed that summarizes all findings.

Acknowledgments

We would like to express our gratitude towards raters two and three, who helped to find a ground truth. We acknowledge the support of Linnaeus University's Centre for Data Intensive Sciences and Applications (DISA) and the Swedish Research School of Management and IT (MIT). This work was supported by the European structural and investment funds (ESIF) project CZ.02.1.01/0.0/0.0/17_048/0007267 (InteCom). We express our gratitude towards the anonymous reviewers who provided thorough and constructive feedback that allowed us to considerably improve our work.

References

[1] C.J. Neill, P.A. Laplante, and J.F. DeFranco, *Antipatterns: Managing Software Organizations and People*, 2nd ed., Auerbach Publications, 2011.

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A Pattern Language – Towns, Buildings, Construction*, Oxford University Press, 1977.

[3] W.H. Brown, R.C. Malveau, H.W. McCormick III, and T.J. Mowbray, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley and Sons, Inc., 1998.

[4] P.A. Laplante and C.J. Neill, *Antipatterns: Identification, Refactoring, and Management* (Auerbach Series on Applied Software Engineering), 1st ed., CRC Press, Auerbach Publications, 2005, 336 pp.

[5] L. Simeckova, P. Brada, and P. Picha, "SPEM-based process anti-pattern models for detection in project data," in *46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020, Portoroz, Slovenia, August 26–28, 2020*, IEEE, 2020, pp. 89–92.

[6] I. Stamelos, "Software project management anti-patterns," *Journal of Systems and Software*, Vol. 83, No. 1, pp. 52–59, 2010.

[7] R.R. Nelson, "It project management: Infamous failures, classic mistakes, and best practices," *MIS Quarterly Executive*, Vol. 6, No. 2, 2008. [Online]. Available: https://aisel.aisnet.org/misqe/vol6/iss2/4

[8] R.S. Kenett and E.R. Baker, *Software Process Quality: Management and Control* (Computer Aided Engineering New York, N.Y., 6), 1st ed., Marcel Dekker, Inc., 1999.

[9] C.P. Halvorsen and R. Conradi, "A taxonomy to compare SPI frameworks," in *Software Process Technology, 8th European Workshop, EWSPT 2001, Witten, Germany, June 19–21, 2001, Proceedings*, V. Ambriola, Ed., Ser. Lecture Notes in Computer Science, Vol. 2077, Springer, 2001, pp. 217–235.

[10] A. Birk, T. Dingsoyr, and T. Stalhane, "Postmortem: Never leave a project without it," *IEEE Software*, Vol. 19, No. 3, pp. 43–45, 2002.

[11] W.J. Brown, H.W. McCormick III, and S.W. Thomas, *AntiPatterns in Project Management*, John Wiley and Sons, Inc., 2000.

[12] P. Silva, A.M. Moreno, and L. Peters, "Software project management: Learning from our mistakes," *IEEE Software*, Vol. 32, No. 3, pp. 40–43, 2015.

[13] A. Nizam, "Software project failure process definition," *IEEE Access*, Vol. 10, pp. 34 428–34 441, 2022.

[14] P. Brada and P. Picha, "Software process anti-patterns catalogue," in *Proceedings of the 24th European Conference on Pattern Languages of Programs, EuroPLoP 2019, Irsee, Germany, July 3–7, 2019*, T.B. Sousa, Ed., Ser. EuroPLoP '19, ACM, 2019, 28:1–28:10.

[15] P.G.F. Matsubara, B.F. Gadelha, I. Steinmacher, and T.U. Conte, "SEXTAMT: A systematic map to navigate the wide seas of factors affecting expert judgment software estimates," *Journal of Systems and Software*, p. 111 148, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221002429

[16] F.U. Muram, B. Gallina, and L.G. Rodriguez, "Preventing omission of key evidence fallacy in process-based argumentations," in *11th International Conference on the Quality of Information and Communications Technology, QUATIC 2018, Coimbra, Portugal, September 4–7, 2018*, A. Bertolino, V. Amaral, P. Rupino, and M. Vieira, Eds., IEEE Computer Society, 2018, pp. 65–73.

[17] P. Picha, P. Brada, R. Ramsauer, and W. Mauerer, "Towards architect's activity detection through a common model for project pattern analysis," in *2017 IEEE International Conference on Software Architecture Workshops, ICSA Workshops 2017, Gothenburg, Sweden, April 5–7, 2017*, IEEE Computer Society, 2017, pp. 175–178.

[18] P. Picha and P. Brada, "Software process anti-pattern detection in project data," in *Proceedings of the 24th European Conference on Pattern Languages of Programs, EuroPLoP 2019, Irsee, Germany, July 3–7, 2019*, T.B. Sousa, Ed., Ser. EuroPLoP '19, ACM, 2019, 20:1–20:12.

[19] D. Settas, S. Bibi, P. Sfetsos, I. Stamelos, and V.C. Gerogiannis, "Using bayesian belief networks to model software project management antipatterns," in *Fourth International Conference on Software Engineering, Research, Management and Applications (SERA 2006), 9–11 August 2006, Seattle, Washington, USA*, IEEE Computer Society, 2006, pp. 117–124.

[20] D. Settas and I. Stamelos, "Using ontologies to represent software project management antipatterns," in *Proceedings of the Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2007), Boston, Massachusetts, USA, July 9–11, 2007*, Knowledge Systems Institute Graduate School, 2007, pp. 604–609.

[21] M.B. Perkusich, G. Soares, H.O. Almeida, and A. Perkusich, "A procedure to detect problems of processes in software development projects using bayesian networks," *Expert Systems with Applications*, Vol. 42, No. 1, pp. 437–450, 2015.

[22] N.E. Fenton, W. Marsh, M. Neil, P. Cates, S. Forey, and M. Tailor, "Making resource decisions for software projects," in *26th International Conference on Software Engineering (ICSE 2004), May 23–28 2004, Edinburgh, United Kingdom*, A. Finkelstein, J. Estublier, and D.S. Rosenblum, Eds., IEEE Computer Society, 2004, pp. 397–406.

[23] M. Unterkalmsteiner, T. Gorschek, A.M. Islam, C.K. Cheng, R.B. Permadi, and R. Feldt, "Evaluation and measurement of software process improvement – A systematic literature review," *IEEE Transactions on Software Engineering*, Vol. 38, No. 2, pp. 398–424, 2012.

[24] J.J.P. Schalken, S. Brinkkemper, and H. van Vliet, "Using linear regression models to analyse the effect of software process improvement," in *Product-Focused Software Process Improvement, 7th International Conference, PROFES 2006, Amsterdam, The Netherlands, June 12–14,*

2006, Proceedings, J. Münch and M. Vierimaa, Eds., Ser. Lecture Notes in Computer Science, Vol. 4034, Springer, 2006, pp. 234–248.

[25] R.K. Yin, *Case Study Research: Design and Methods* (Applied Social Research Methods), 5th ed., SAGE Publications, 2013.

[26] R.W. Scholz and O. Tietje, *Embedded Case Study Methods: Integrating Quantitative and Qualitative Knowledge*, 1st, SAGE Publications, Inc., 2001.

[27] T. Shaikhina, D. Lowe, S. Daga, D. Briggs, R. Higgins, and N. Khovanova, "Machine learning for predictive modelling based on small data in biomedical engineering," *IFAC-PapersOnLine*, Vol. 48, No. 20, pp. 469–474, 2015, 9th IFAC Symposium on Biological and Medical Systems BMS 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896315020765

[28] Y. Zhang and C. Ling, "A strategy to apply machine learning to small datasets in materials science," *npj Computational Materials*, Vol. 4, No. 1, p. 25, May 2018.

[29] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering – Guidelines and Examples*, Wiley, 2012. [Online]. Available: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118104358.html

[30] K.L. Gwet, *Handbook of Inter-Rater Reliability: The Definitive Guide to Measuring the Extent of Agreement Among Raters*, 4th ed., Advanced Analytics, Sep. 2014.

[31] E. Tüzün, H. Erdogmus, M.T. Baldassarre, M. Felderer, R. Feldt, and B. Turhan, "Ground--truth deficiencies in software engineering: When codifying the past can be counterproductive," *IEEE Software*, Vol. 39, No. 3, pp. 85–95, 2022.

[32] O. Bousquet and A. Elisseeff, "Stability and generalization," *Journal of Machine Learning Research*, Vol. 2, pp. 499–526, 2002. [Online]. Available: http://jmlr.org/papers/v2/bousquet02a.html

[33] K. Scott, *The unified process explained*, 1st ed., Boston, MA: Addison Wesley Professional, Nov. 2001.

[34] P. Kroll and P. Kruchten, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP* (Addison-Wesley object technology series), Boston, MA: Addison-Wesley Educational, Apr. 2003, 464 pp.

[35] W.R. Shadish, T.D. Cook, and D.T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*, 3rd ed., Houghton Mifflin Company, 2002.

[36] J.M. Verner, J. Sampson, V. Tosic, N.A.A. Bakar, and B.A. Kitchenham, "Guidelines for industrially-based multiple case studies in software engineering," in *Proceedings of the Third IEEE International Conference on Research Challenges in Information Science, RCIS 2009, Fès, Morocco, April 22–24, 2009*, A. Flory and M. Collard, Eds., IEEE, 2009, pp. 313–324.

[37] R. Zhu, D. Zeng, and M.R. Kosorok, "Reinforcement learning trees," *Journal of the American Statistical Association*, Vol. 110, No. 512, pp. 1770–1784, 2015, PMID:26903687.

[38] D. Draheim and L. Pekacki, "Process-centric analytical processing of version control data," in *6th International Workshop on Principles of Software Evolution (IWPSE 2003), September 1–2, 2003, Helsinki, Finland*, IEEE Computer Society, 2003, p. 131.

[39] R. Ramsauer, D. Lohmann, and W. Mauerer, "Observing custom software modifications: A quantitative approach of tracking the evolution of patch stacks," in *Proceedings of the 12th International Symposium on Open Collaboration, OpenSym 2016, Berlin, Germany, August 17–19, 2016*, A.I. Wasserman, Ed., ACM, 2016, 4:1–4:4.

[40] D.A. Tamburri, F. Palomba, A. Serebrenik, and A. Zaidman, "Discovering community patterns in open-source: A systematic approach and its evaluation," *Empirical Software Engineering*, Vol. 24, No. 3, pp. 1369–1417, 2019.

[41] S.Z. Talpová and T. Čtvrtníková, "Scrum anti-patterns, team performance and responsibility," *International Journal of Agile Systems and Management*, Vol. 14, No. 1, p. 170, 2021.

[42] A. Hachemi, "Software development process modeling with patterns," in *WSSE 2020: The 2nd World Symposium on Software Engineering, Chengdu, China, September 25–27, 2020*, ACM, 2020, pp. 37–41.

[43] T. Frtala and V. Vranic, "Animating organizational patterns," in *8th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2015, Florence,*

Italy, May 18, 2015, A. Begel, R. Prikladnicki, Y. Dittrich, C.R.B. de Souza, A. Sarma, and S. Athavale, Eds., IEEE Computer Society, 2015, pp. 8–14.

[44] A.H.M. ter Hofstede, C. Ouyang, M.L. Rosa, L. Song, J. Wang, and A. Polyvyanyy, "APQL: A process-model query language," in *Asia Pacific Business Process Management – First Asia Pacific Conference, AP-BPM 2013, Beijing, China, August 29–30, 2013. Selected Papers*, M. Song, M.T. Wynn, and J. Liu, Eds., Ser. Lecture Notes in Business Information Processing, Vol. 159, Springer, 2013, pp. 23–38.

[45] J. Roa, E. Reynares, M.L. Caliusco, and P.D. Villarreal, "Towards ontology-based anti-patterns for the verification of business process behavior," in *New Advances in Information Systems and Technologies – Volume 2 [WorldCIST'16, Recife, Pernambuco, Brazil, March 22–24, 2016]*, Ser. Advances in Intelligent Systems and Computing, Á. Rocha, A.M.R. Correia, H. Adeli, L.P. Reis, and M.M. Teixeira, Eds., Vol. 445, Springer, 2016, pp. 665–673.

[46] A. Awad, A. Barnawi, A. Elgammal, R.E. Shawi, A. Almalaise, and S. Sakr, "Runtime detection of business process compliance violations: An approach based on anti patterns," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13–17, 2015*, R.L. Wainwright, J.M. Corchado, A. Bechini, and J. Hong, Eds., ACM, 2015, pp. 1203–1210.

[47] T.O.A. Lehtinen, M. Mäntylä, J. Vanhanen, J. Itkonen, and C. Lassenius, "Perceived causes of software project failures – an analysis of their relationships," *Information and Software Technology*, Vol. 56, No. 6, pp. 623–643, 2014.

[48] L. Rising and N.S. Janoff, "The scrum software development process for small teams," *IEEE Software*, Vol. 17, No. 4, pp. 26–32, 2000.

[49] P.G. Smith and D.G. Reinertsen, *Developing Products in Half the Time: New Rules, New Tools*, 2nd ed., Nashville, TN: John Wiley and Sons, Oct. 1997.

[50] F.P. Brooks Jr, *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*, 2nd ed., Boston, MA: Addison-Wesley Longman, Aug. 1995.

[51] P. Picha et al., "Process anti-pattern detection in student projects – a case study," in *Proceedings of the 27th European Conference on Pattern Languages of Programs, EuroPLoP 2022, Irsee, Germany, July 6–10, 2022*, T.B. Sousa, Ed., Ser. EuroPLoP '22, ACM, 2022.

[52] E.B. Swanson, "The dimensions of maintenance," in *Proceedings of the 2nd International Conference on Software Engineering, San Francisco, California, USA, October 13–15, 1976*, R.T. Yeh and C.V. Ramamoorthy, Eds., IEEE Computer Society, 1976, pp. 492–497. [Online]. Available: http://dl.acm.org/citation.cfm?id=807723

[53] S. Hönel, M. Ericsson, W. Löwe, and A. Wingkvist, "Using source code density to improve the accuracy of automatic commit classification into maintenance activities," *Journal of Systems and Software*, Vol. 168, p. 110 673, 2020.

[54] D.I.K. Sjøberg, T. Dybå, B.C.D. Anda, and J.E. Hannay, "Building theories in software engineering," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D.I.K. Sjøberg, Eds., Springer, 2008, pp. 312–336.

[55] C. Wohlin and A. Rainer, "Is it a case study? – A critical analysis and guidance," *Journal of Systems and Software*, Vol. 192, p. 111 395, 2022.

[56] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, 1st ed., Springer, 2012.

[57] S. Hönel and C. Wohlin, Personal communication, Prof. Wohlin recently authored guidelines for correctly classifying studies [55]., Dec. 2022.

[58] M.J. Tiedeman, "Post-mortems – Methodology and experiences," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 2, pp. 176–180, 1990.

[59] B. Collier, T. DeMarco, and P. Fearey, "A defined process for project post mortem review," *IEEE Software*, Vol. 13, No. 4, pp. 65–72, 1996.

[60] J. Gerring, *Case Study Research: Principles and Practices* (Strategies for Social Inquiry), 2nd ed., Cambridge University Press, 2017.

[61] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *Proceedings of the Third International Symposium on Empirical Software Engineering and*

Measurement, ESEM 2009, October 15–16, 2009, Lake Buena Vista, Florida, USA, IEEE Computer Society, 2009, pp. 401–404.

[62] S. Hönel, P. Pícha, P. Brada, L. Rychtarova, and J. Danek, *Detection of the Fire Drill anti-pattern: 15 real-world projects with ground truth, issue-tracking data, source code density, models and code*, The repository for the source code based method is at: https://github.com/MrShoenel/anti-pattern-models, Jan. 2023.

[63] D. Chappell, *What is application lifecycle management?* Dec. 2008. [Online]. Available: https://web.archive.org/web/20141207012857/http://www.microsoft.com/global/applicationplatform/en/us/RenderingAssets/Whitepapers/What%20is%20Application%20Lifecycle%20Management.pdf [Acessed: 07. 12. 2014].

[64] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, Vol. 14, No. 2, pp. 131–164, 2009.

[65] J. Cohen, "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit.," *Psychological bulletin*, Vol. 70, No. 4, p. 213, 1968.

[66] K.L. Gwet, "Computing inter-rater reliability and its variance in the presence of high agreement," *British Journal of Mathematical and Statistical Psychology*, Vol. 61, No. 1, pp. 29–48, 2008.

[67] J.R. Landis and G.G. Koch, "An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers," *Biometrics*, Vol. 33, No. 2, pp. 363–374, Jun. 1977, PMID:884196.

[68] D. Klein, "Implementing a general framework for assessing interrater agreement in stata," *The Stata Journal*, Vol. 18, No. 4, pp. 871–901, 2018.

[69] B.W. Boehm, *Software Engineering Economics*, 1st ed., Philadelphia, PA: Prentice Hall, Oct. 1981.

[70] N.C. Dalkey, "The Delphi Method: An experimental study of group opinion," The RAND Corporation, Santa Monica, CA, Tech. Rep., 1969, Document Number: RM-5888-PR. [Online]. Available: https://www.rand.org/pubs/research_memoranda/RM5888.html

[71] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, Vol. 27, No. 3, pp. 832–837, 1956, zbMATH:0073.14602, MathSciNet:MR79873.

[72] D.M. Endres and J.E. Schindelin, "A new metric for probability distributions," *IEEE Transactions on Information Theory*, Vol. 49, No. 7, pp. 1858–1860, 2003.

[73] B. Hofner, L. Boccuto, and M. Göker, "Controlling false discoveries in high-dimensional situations: Boosting with stability selection," *BMC Bioinformatics*, Vol. 16, No. 1, p. 144, May 2015.

[74] W.N. Venables and B.D. Ripley, *Modern Applied Statistics with S*, Fourth, New York: Springer, 2002. [Online]. Available: http://www.stats.ox.ac.uk/pub/MASS4

[75] F. Bertrand and M. Maumy-Bertrand, *plsRglm: Partial least squares linear and generalized linear regression for processing incomplete datasets by cross-validation and bootstrap techniques with R*, arXiv, 2018.

[76] A. Peters and T. Hothorn, *ipred: Improved Predictors*, R package version 0.9-9, 2019. [Online]. Available: https://CRAN.R-project.org/package=ipred

[77] F. Pukelsheim, "The three sigma rule," *The American Statistician*, Vol. 48, No. 2, pp. 88–91, 1994. [Online]. Available: http://www.jstor.org/stable/2684253

[78] S. Hönel, "Technical reports compilation: Detecting the Fire Drill anti-pattern using source code and issue-tracking data," *CoRR*, Vol. abs/2104.15090, Jan. 2023.

[79] D. Vysochanskij and Y.I. Petunin, "Justification of the 3σ rule for unimodal distributions," *Theory of Probability and Mathematical Statistics*, Vol. 21, pp. 25–36, 1980.

[80] P. Tchébychef, "Des Valeurs Moyennes," *Journal de Mathématiques Pures et Appliquées*, 2nd Ser., Vol. 12, pp. 177–184, 1867, Traduction du Russe par M. N. de Khanikof. [Online]. Available: http://eudml.org/doc/234989

[81] G.C. Cawley and N.L.C. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, Vol. 11, pp. 2079–2107, 2010.

[82] S. Raudys and A.K. Jain, "Small sample size effects in statistical pattern recognition: Recommendations for practitioners," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, No. 3, pp. 252–264, 1991.

[83] S. Varma and R. Simon, "Bias in error estimation when using cross-validation for model selection," *BMC Bioinformatics*, Vol. 7, No. 1, Feb. 2006.

[84] A. Vabalas, E. Gowen, E. Poliakoff, and A.J. Casson, "Machine learning algorithm validation with a limited sample size," *PLOS ONE*, Vol. 14, No. 11, E. Hernandez-Lemus, Ed., pp. 1–20, Nov. 2019.

[85] T. Shaikhina, D. Lowe, S. Daga, D. Briggs, R. Higgins, and N. Khovanova, "Machine learning for predictive modelling based on small data in biomedical engineering," *IFAC-PapersOnLine*, Vol. 48, No. 20, pp. 469–474, 2015.

[86] L. Torgo, R.P. Ribeiro, B. Pfahringer, and P. Branco, "SMOTE for regression," in *Progress in Artificial Intelligence – 16th Portuguese Conference on Artificial Intelligence, EPIA 2013, Angra do Heroísmo, Azores, Portugal, September 9–12, 2013. Proceedings*, L. Correia, L.P. Reis, and J. Cascalho, Eds., Ser. Lecture Notes in Computer Science, Vol. 8154, Springer, 2013, pp. 378–389.

[87] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2020. [Online]. Available: https://www.R-project.org/

[88] B. Greenwell, B. Boehmke, J. Cunningham, and G. Developers, *gbm: Generalized Boosted Regression Models*, R package version 2.1.8, 2020. [Online]. Available: https://CRAN.R-project.org/package=gbm

[89] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, Vol. 2, No. 3, pp. 18–22, 2002. [Online]. Available: https://CRAN.R-project.org/doc/Rnews/

[90] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis, "Kernlab – an S4 package for kernel methods in R," *Journal of Statistical Software*, Vol. 11, No. 9, pp. 1–20, 2004. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v011i09

[91] P.A. Lachenbruch and M.R. Mickey, "Estimation of error rates in discriminant analysis," *Technometrics*, Vol. 10, No. 1, pp. 1–11, 1968. [Online]. Available: http://www.jstor.org/stable/1266219

[92] M.J. Kearns and D. Ron, "Algorithmic stability and sanity-check bounds for leave-one-out cross-validation," *Neural Computation*, Vol. 11, No. 6, pp. 1427–1453, 1999.

[93] J.L. Fleiss, J. Cohen, and B.S. Everitt, "Large sample standard errors of kappa and weighted kappa.," *Psychological Bulletin*, Vol. 72, No. 5, pp. 323–327, Nov. 1969.

[94] D.V. Cicchetti and S.A. Sparrow, "Developing criteria for establishing interrater reliability of specific items: applications to assessment of adaptive behavior," *American Journal of Mental Deficiency*, Vol. 86, No. 2, pp. 127–137, Sep. 1981.

[95] J.L. Fleiss, *Statistical Methods for Rates and Proportions* (Probability and Mathematical Statistics S.), 2nd ed., Nashville, TN: John Wiley and Sons, May 1981.

[96] D.A. Regier et al., "DSM-5 field trials in the United States and Canada, part II: Test-retest reliability of selected categorical diagnoses," *American Journal of Psychiatry*, Vol. 170, No. 1, pp. 59–70, Jan. 2013.

[97] A.S. Lee, "A scientific methodology for MIS case studies," *MIS Quarterly*, Vol. 13, No. 1, pp. 33–50, 1989.

Appendices

The following appendices provide additional information about the Fire Drill phenomenon in full (A), the projects' setup (B), the Fire Drill's observed symptoms and consequences, as well as supercategories (C), observations counter-indicative of a Fire Drill (D), and, lastly, a more detailed and numeric view of the quantitative analysis' variable importance (E).

Appendix A. Full Fire Drill description

Here, we include the most recent and complete description of the Fire Drill anti-pattern using a pattern language and a typically structured template [3, 12]. This study operates on this description and any won insights, new results, forces, symptoms and consequences, etc., were *not* incorporated into this original description. The following is an exact copy of the original resource by Picha et al. [51][11]. The elements *Also Known As*, *Variations (optional)*, *Example(s) (optional)*, and *Notes (optional)* were left out as they are currently empty and reserved for future use.

Fire Drill

Summary

Requirements and Analysis phases prolonged and consuming disproportionate amount of resources (because management want to do them "right"), then frantic "everything needs to be done yesterday" period to finish on time (when management finds out they wasted most of project's schedule and resources on analysis).

Context

Waterfall(ish) projects, especially when project oversight is loose and/or management is not driven by outcome.

Unbalanced forces

- need (desire) to have specifications perfect,
- management consumed by internal (political) issues,
- actual development of a high-quality product takes time,
- quality objectives formally stated and high,
- strict deadlines for delivery.

Symptoms and consequences

- long period at project start where activities connected to requirements, analysis and planning prevail, and design and implementation activities are rare,
- only analytical or documentational artefacts for a long time,

[11] Full Fire Drill Description. 2022. https://github.com/ReliSA/Software-process-antipatterns-catalogue/blob/master/catalogue/Fire_Drill.md

– relatively short period towards project end with sudden increase in development efforts (i.e. rock-edge burndown, especially when viewing implementation tasks only),
– little testing/QA and project progress tracking activities during development period,
– final product with poor code quality, many open bug reports, poor or patchy documentation,
– if the previous trhee points do not apply, project schedule or scope is compromised (i.e., either delayed delivery or descoping occurs),
– stark contrast between interlevel communication in project hierarchy (management - developers) during the first period (close to silence) and after realizating the problem (panic and loud noise).

Causes

– management does not take seriously development effort (time) estimates,
– management absorbed in "various technopolitical issues (…) prevent[ing] the development staff from making progress",
– team is happy to produce artifacts early in the project,
– requirements are complex and their prioritization is not forced early on,
– team overseeing the need to prioritize "working code over comprehensive documentation",
– management wants to appear the project to be on track,
– management believes it is more important to deliver complete functionality than good quality,
– project tracking and oversight is loose, easily lulled inco complacency by easy-to-reach outcomes.

(Refactored) solution

– force the team to start delivering (parts of) the "consumable solution" early, possibly alongside the analysis and planning artefacts, by instituting strong project tracking and oversight related to actual outcomes,
– it helps to follow an iterative process, architecture-driven development, and have a well-performing product owner.

Anti-pattern	Relation
Analysis Paralysis	potential cause
Collective Procrastination [18]	more generic case

Sources

[12], [SOU' 18], Fire Drill, [3]

Appendix B. Project setup

Table B1 gives an overview about each project's setup, including number of team members, project duration, number of iterations, man-hours logged on issues, and the number of issues or commits per activity (see Subsection 4.5). Each project developed a different kind

Table B1. General characteristics of the student projects

Project	Team members	Project duration (d)	Iterations	Time spent (h)	Issues	Time logged (h) REQ/DEV/DESC	Commits A/C/P
1	4	78	6	277.95	104	62/158/0	36/32/48
2	4	93	6	399.35	98	115/177.9/4	42/108/76
3	4	97	7	303.80	113	59.3/136.7/24.5	26/35/50
4	4	100	6	346.75	118	67.5/221.8/66	29/59/38
5	2	61	4	238.25	44	60/131.3/14	33/26/51
6	4	98	5	234.62	101	39.4/157.9/2	79/63/75
7	5	100	6	396.35	114	80.8/183.1/26.5	83/64/36
8	4	94	7	206.00	85	46.5/89.5/0	10/6/14
9	4	84	6	285.35	112	37/176.3/51.5	54/23/23
10	4	91	6	348.40	128	59.7/152.8/9.5	74/38/46
11	4	83	6	300.00	114	45.8/147.6/15	151/168/223
12	4	94	7	292.25	81	35/172.6/0	35/61/40
13	4	105	6	409.25	181	49.3/210.6/122.4	67/48/77
14	3	95	6	204.75	72	14.3/104.8/7.4	34/8/20
15	4	98	6	246.30	62	18.8/86.5/0.8	24/27/15

of application. The following list gives a brief description for the developed application in each.

Pr. 1: Desktop and web application for full-text searches in scanned documents
Pr. 2: Heatmap web application over open data
Pr. 3: Enhancement of web application for complex graph visualization
Pr. 4: Web application for HR management
Pr. 5: A Universal deserializer in and for Java
Pr. 6: Upgrade of a web application for linguistic research
Pr. 7: Mobile application for museum visitors
Pr. 8: Bitmap generator for public transportation
Pr. 9: Web application simulating pivot tables
Pr. 10: Client web front-end for sensor data
Pr. 11: Application for certificate management with web and desktop interfaces
Pr. 12: Web application over two databases with linguistic research data
Pr. 13: Web and mobile application for open weather data visualization and prediction
Pr. 14: Virtual Reality application arm rehabilitation machine control
Pr. 15: Sensor dashboard for the Raspberry Pi platform

Lastly, the projects were free to choose a programming language that best suited their needs and conformed with the customer's requirements. The following list shows the share each language had, obtained from the lines of code[12].

Pr. 1: Desktop and web application for full-text searches in s
Pr. 1: 43.64 % Java, 37.90 % TypeScript, 9.70 % HTML, and 8.76 % Others
Pr. 2: 30.90 % Python, 18.28 % CSS, 18.23 % PHP, 11.41 % SCSS, 8.50 % JavaScript, 8.48 % Twig, and 4.19 % Others
Pr. 3: 57.89 % JavaScript, 28.16 % Java, 8.47 % CSS, and 5.48 % Others
Pr. 4: 77.41 % Java, 17.29 % JavaScript, and 5.30 % Others
Pr. 5: 98.95 % JavaScript, and 1.05 % Java and Others
Pr. 6: 88.34 % PHP, 3.29 % PLpgSQL, and 8.37 % HTML, JavaScript and Others
Pr. 7: 63.71 % PHP, 35.48 % Blade, and 0.81 % Others
Pr. 8: 100.00 % Java

[12]GitHub Linguist was used to compute the shares, see https://github.com/github/linguist (2023).

Pr. 9: 78.84 % Java, 10.92 % HTML, 7.84 % HTML, and 2.39 % CSS
Pr. 10: 62.87 % TypeScript, 24.07 % SCSS, 12.03 % HTML, and 1.03 % Others
Pr. 11: 80.14 % Python, 10.57 % HTML, 7.32 % JavaScript, and 1.97 % Others
Pr. 12: 42.52 % JavaScript, 27.30 % HTML, 16.58 % PHP, 11.17 % Java, and 2.43 % CSS and Others
Pr. 13: 72.51 % C#, 22.92 % ShaderLab, 3.76 % HLSL, and 0.82 % HTML
Pr. 14: 76.31 % C++, 14.44 % C#, 6.37 % Python, 1.79 % QMake, and 1.09 % Others
Pr. 15: 100.00 % Python

Appendix C. Fire Drill symptoms and consequences

This appendix shows the results of the systematic analysis of the raters' notes for each project, according to the methodology described in Subsection 4.4. The raters' notes, as well as all other data is to be found in [62]. This list is an excerpt from the most recent Fire Drill description[13]. It is organized into a top-level list of symptoms and consequences (`SC1`–`SC7` and `ESC1`–`ESC3`) and a nested list for each with concrete observed, empirical instances (denoted as `Exx`). Each empirical instance has a severity attached. It follows the format `[project, rater(s), severity]`, e.g., `[1,AB,3]` indicates that raters A and B commonly identify a problem instance in project one, and the severity is three out of five. As of the second pass, some observations and their severity were aggregated. For example, (`[13,A,3]`, `[13,B,2]`, `[13,C,4]`) ⇒`[13,ABC,5]` means that all three raters observed a symptom/consequence in project 13, with varying severity each. The aggregation reduces this to a single observation with higher severity, according to the ordinal scale's description.

- `SC1`. long period at project start where activities connected to requirements, analysis and planning prevail, and design and implementation activities are rare,
  - `E01`. can be caused by, e.g., the team needing some time for familiarization with the (new, changed) tools, way of communication, or process in the beginning `[1,A,1]`, `[3,B,2]`, `[5,A,1]`, `[7,B,1]`, `[9,B,2]`, `[13,A,1]`, `[14,A,1]`, `[15,A,2]`,
  - `E02`. work delayed due to external factors, such as ramping-down of previous project or other, non-related work `[1,A,0]`, `[15,A,0]`,
  - `E03`. project is under- or over-scoped from the beginning, so the team spends time idling or does not know where to start `[3,C,1]`, (`[13,A,3]`, `[13,B,2]`, `[13,C,4]`) ⇒`[13,ABC,5]`, `[15,C,1]`;
- `SC2`. only analytical or documentational artifacts for a long time,
  - `E04`. Development takes place only at the end of an iteration, sometimes rushed (opposite of `CISC08`) `[3,C,1]`, `[6,A,1]`, `[14,A,3]`;
- `SC3`. relatively short period towards project end with a sudden increase in development efforts (i.e. rock-edge burndown, especially when viewing implementation tasks only),
  - `E05`. team rushes to deliver at least an MVP to meet the final deadline `[13,A,3]`;
- `SC4`. little testing/QA and project progress tracking activities during the development period,
  - `E06`. sometimes caused by improper usage of project management tools, for example logging time only at the end of a phase `[3,C,2]`,

[13]The Fire Drill description in the process anti-pattern catalog. 2023. https://github.com/Mr-Shoenel/Software-process-antipatterns-catalogue/blob/7a4d8/catalogue/Fire_Drill.md.

  - E07. too much focus on "visible" progress by managerial decision while testing/QA is neglected, which leads to an accumulation of technical debt in the long-term (Half Done Is Enough) [12,A,1];
- SC5. final product with poor code quality, many open bug reports, poor or patchy documentation,
  - E08. too meet the final delivery date, the product quality is decreased by skipping, e.g., features or proper Q/A (alternatively, the may be delivered late, but as agreed) ([4,B,1], [4,C,2]) ⇒[4,BC,3], [6,C,2], [7,B,2], ([13,A,4], [13,C,2]) ⇒[13,AC,5],
- SC6. if points SC3 through SC5 do not apply, (likely) the project schedule or scope is compromised (i.e., either delayed delivery or descoping occurs),
  - E09. team accepts change requests, re-prioritization of existing or new issues within a phase (e.g., after the start of a sprint); improper change management process [1,A,1], [3,C,4], ([5,A,1], [5,B,1]) ⇒[5,AB,2], [9,A,1], [10,B,2], [12,A,3], ([13,A,4], [13,B,4]) ⇒[13,AB,5],
  - E10. planned work is not completed and overflows into the next phase (e.g., sprint), due to, e.g., an over-challenged team (opposite of CISC28), misestimation, or unequal work distribution [3,C,4], [7,C,3], [9,C,3], [10,C,1], [14,A,1], [15,B,1],
  - E11. iterations are too short and are artificially prolonged, forcing the team to do overtime or to truncate the workload [3,C,4], [4,C,2], [7,C,1];
- SC7. stark contrast between interlevel communication in project hierarchy (management – developers) during the first period (close to silence) and after realizing the problem (panic and loud noise).

Here is a list of new, empirical symptoms and causes:

- ESC1. poor communication (e.g., unresponsive, relayed, large overhead, or underqualified decision-maker) between stakeholders (e.g., customer) and the development team,
  - E12. unresponsive customer or unsatisfactory (e.g., late, incomplete, or slow) communication (critical infos or materials not provided timely) ([3,A,4], [3,B,4]) ⇒[3,AB,5], ([4,A,3], [4,B,4], [4,C,3]) ⇒[4,ABC,5], [5,C,1], ([6,A,3], [6,B,3]) ⇒[6,AB,4], ([7,A,3], [7,C,3]) ⇒[7,AC,4], ([10,A,3], [10,B,3], [10,C,2]) ⇒[10,ABC,5], [11,A,1], [12,A,1], [13,C,1], [14,A,1], ([15,A,1], [15,C,1]) ⇒[15,AC,2],
  - E13. requirements cannot be clearly negotiated or are ambiguous [3,C,3], ([10,A,3], [10,B,3]) ⇒[10,AB,4], ([13,A,3], [13,C,4]) ⇒[13,AC,5],
  - E14. post-negotiation misunderstandings (without proper re-negotiation) [3,A,2],
  - E15. tacit misunderstanding (stakeholder and team believe they are on the same page, but they are not in actuality) [3,A,3], [4,C,1],
  - E16. customer interferes with project management without properly communicating the made changes, which directly translates into a project risk [9,B,2];
- ESC2. high project risk (opposite of CISC26), often manifests itself through, e.g., unrealistic work item estimates, the absence of proper testing (opposite of CISC09), or improper documentation,
  - E17. business requirements (tacitly/unknowingly) misinterpreted ([3,C,3], [3,A,3]) ⇒[3,AC,4], [4,C,3], [6,A,2], [12,A,1],
  - E18. finalized work does not conform to the defined specification/expectation [3,C,2],
  - E19. new functionality introduces bugs, and not enough slack was allocated during planning for the fixing (or preventing by proper testing) of these [3,C,1], [4,A,2],

  - E20. tasks are done in the wrong order (Cart Before Horse), esp. development before properly analyzing and planning ([6,A,2], [6,B,2]) ⇒[6,AB,3], [12,A,1], [15,A,1]
  - E21. imbalanced activities at the beginning, end, or during the project, such as too much focus on development early and requirements analysis later that leads to descoping, for example ([6,B,2], [6,C,2]) ⇒[6,BC,3], ([7,A,1], [7,B,1]) ⇒[7,AB,2], [9,A,1], [12,C,1], ([13,A,3], [13,B,2]) ⇒[13,AB,4],
  - E22. lack of experience that leads to misestimation of work items [6,C,1], [9,A,1], [13,C,2],
  - E23. work items or goals not properly defined, absent, or defined too late [7,C,4], [13,A,4],
  - E24. management fails to ascertain that the development team is available to its planned capacity (e.g., it allows the team to be affected by external factors), which has a negative impact on the progression of the project, such as descoping, quality regression, or delayed delivery [9,C,1], [11,A,1], [13,A,1],
  - E25. Strong dependency between stakeholders and the development team, such that the team cannot proceed very long or at all by themselves (opposite of CISC15) [10,C,1],
  - E26. technical difficulties in the environment, such as the infrastructure, which cause unexpected delays to, e.g., the development or deployment ([12,A,3], [12,B,1]) ⇒[12,AB,3], ([15,A,1], [15,C,1]) ⇒[15,AC,2],
  - E27. frequent project schedule adaptions, manifested by excessive use of administrative tools, leading to a low-quality product [13,B,4], [15,A,2];
- ESC3. poor usage of project management tools and methodologies which gives rise to management misinterpreting the progress and state of the project,
  - E28. too-large goals that were not properly broken down into smaller issues [3,C,1],
  - E29. mislabeling of items; for example, marking an Epic as a Task [4,C,1],
  - E30. a discrepancy between the defined work and the actual work exists, due to, e.g., time not logged properly, issues not defined, or work completed during undocumented overtime [3,C,1], [7,C,4], [10,A,1], [15,A,1],
  - E31. information mismanagement, for example, by duplication or using too many different systems for storing and disseminating information [4,B,1].

Appendix D. Symptoms and consequences indicating the absence

During the analysis of the raters' notes, recurring elements of healthy projects, showing no or miniscule signs of a Fire Drill, emerged. While the absence of evidence does not mean that a Fire Drill is not present, we gathered some empirical evidence for symptoms and consequences that would be *counter-indicative* of the phenomenon. The following unordered list is another excerpt from the most recent description. It is numbered similarly to the list of symptoms and consequences (CISC01–CISC29), but does not include the number of observations or how strong an observation manifested. We use the abbreviation CISC to mean counter-indicative symptom and consequence.

- CISC01. communication and collaboration with the customer is seamless;
- CISC02. no descoping which typically happens towards the end of a phase (sprint, milestone, etc.);
- CISC03. timely product delivery according to agreed-upon quality;

- CISC04. satisfaction among all stakeholders (e.g., team, customer, etc.);
- CISC05. regular and successful iteration evaluations that do not result in the unveiling of (large/additional) problems;
- CISC06. clear understanding of the requirements and resulting unproblematic execution;
- CISC07. equal (or almost equal) work distribution among team members (also: fair work distribution among differently-skilled/-tasked team members);
- CISC08. linear burn-down (i.e., done work is distributed uniformly, instead of, e.g., at the end of a phase);
- CISC09. product tested properly (e.g., appropriate tests and/or good coverage), as well regularly/continuously;
- CISC10. starting to implement features right from the project inception (clear requirements);
- CISC11. proper allocation of project resources (esp. time);
- CISC12. proper (planning of) distribution of time (spent) across the required activities (e.g., enough time spent on defining requirements properly);
- CISC13. appropriate prioritization of activities when resources (often time) become (temporarily) scarce;
- CISC14. successful intermediate and final product deliveries (according to customer's acceptance criteria);
- CISC15. team can proceed at least short-term even if the customer is unavailable (good internal crisis management);
- CISC16. accurate work-item estimates (time, points, etc.), esp. no over-estimation (which indicates high level of uncertainty and, therefore, risk);
- CISC17. project management tool(s) used accordingly; e.g., proper usage of primitives (item types), the Scrum/Kanban board (or swimlanes), regular updates (all these indicate proper management);
- CISC18. regular activities according to used methodology (e.g., Scrum), such as daily meetings, retrospectives, and milestones;
- CISC19. change requests, re-prioritization of existing or new issues are rejected by the team once the phase (e.g., sprint) started in which they were planned for (as should be);
- CISC20. proper communication among team members; direct messaging, as well as dedicated channels and often the usage of bots (from, e.g., a CI pipeline);
- CISC21. efficient communication with customer, that is, direct (no relays), quick, unproblematic, of high quality, tending to the necessary aspects of the product (low overhead);
- CISC22. stable team (no developer churn) and harmony among members;
- CISC23. mutual understanding: effort estimations between customer and team are similar (customer understands technical challenges and team understands business requirements);
- CISC24. activities in right order (e.g., analysis before design before implementation etc.);
- CISC25. progress is reflected empirically (objectively), i.e., provably no discrepancy between reported and actual progress exists;
- CISC26. proper risk management through, e.g., the development of prototypes;
- CISC27. the scope may change and adapt over the course of the project (due to the agile nature), but it does not increase/widen without additional resources;

– `CISC28`. team is not undersized for the project: no (steady) accumulation of non-finished work items into the next phase;
– `CISC29`. when external forces and (un)forseeable events happen, development is suspended and/or the product is delayed accordingly, allowing the team to catch up (rather than forcing them to do overtime).

Appendix E. Detailed variable importance

Tables E1 and E2 show the detailed variable importance for source code and issue-tracking, respectively. Both tables reflect the same result as shown in Figure 7.

Table E1. Detailed variable importance (scaled to percent) for the no-pattern source code dataset, including means and sums across segments and features

Variable (feature)	Seg. 1	Seg. 2	Seg. 3	Seg. 4	Seg. 5	Seg. 6	Seg. 7	Seg. 8	Seg. 9	Seg. 10	Sum
A	1.58	0.69	1.44	2.70	1.47	1.67	0.73	0.81	1.44	1.68	14.20
CP	1.53	1.32	3.39	2.19	1.33	2.31	1.38	1.89	1.10	0.72	17.16
$\mathsf{FREQ}_{\mathsf{nm}}$	2.60	1.00	2.35	1.69	0.79	2.62	1.23	1.17	1.59	1.08	16.12
A |0 CP	1.27	1.28	2.87	2.95	2.08	0.59	2.01	3.65	1.13	1.18	19.00
A |0 $\mathsf{FREQ}_{\mathsf{nm}}$	0.64	0.74	2.12	2.27	2.53	2.47	1.61	2.91	1.49	0.77	17.55
CP |0 $\mathsf{FREQ}_{\mathsf{nm}}$	2.65	1.31	3.16	3.02	0.91	1.28	0.98	1.41	0.45	0.80	15.97
Mean	1.71	1.06	2.55	2.47	1.52	1.82	1.32	1.97	1.20	1.04	n.a.
Sum	10.28	6.35	15.33	14.81	9.10	10.94	7.93	11.84	7.21	6.23	200.00

Table E2. Detailed variable importance (scaled to percent) for the no-pattern issue-tracking dataset, including means and sums across segments and features

Variable (Feature)	Seg. 1	Seg. 2	Seg. 3	Seg. 4	Seg. 5	Seg. 6	Seg. 7	Seg. 8	Seg. 9	Seg. 10	Sum
REQ	1.19	1.20	1.50	2.42	1.63	1.25	0.89	0.64	1.15	1.16	13.02
DEV	1.20	2.93	1.70	1.02	1.00	1.55	0.62	1.74	2.48	2.39	16.63
DESC	1.39	0.72	1.24	1.26	0.78	1.50	0.60	2.84	2.47	1.39	14.19
REQ |0 DEV	1.76	1.21	3.14	1.07	1.99	1.68	1.06	2.15	2.91	1.58	18.55
REQ |0 DESC	3.86	2.99	2.94	1.31	1.42	2.07	1.17	1.75	2.23	0.99	20.72
DEV |0 DESC	1.25	1.00	1.04	1.30	2.88	2.68	0.83	2.67	2.23	1.00	16.89
Mean	1.78	1.68	1.93	1.40	1.62	1.79	0.86	1.97	2.24	1.42	n.a.
Sum	10.65	10.05	11.56	8.38	9.69	10.74	5.17	11.79	13.47	8.50	200.00

e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240107, DOI: 10.37190/e-Inf240107

Boosting and Comparing Performance of Machine Learning Classifiers with Meta-heuristic Techniques to Detect Code Smell

Shivani Jain*, Anju Saha*
*Information Technology, GGS Indraprastha University
shivani.1091@gmail.com, anju_kochhar@yahoo.com

Abstract

Background: Continuous modifications, suboptimal software design practices, and stringent project deadlines contribute to the proliferation of code smells. Detecting and refactoring these code smells are pivotal to maintaining complex and essential software systems. Neglecting them may lead to future software defects, rendering systems challenging to maintain, and eventually obsolete. Supervised machine learning techniques have emerged as valuable tools for classifying code smells without needing expert knowledge or fixed threshold values. Further enhancement of classifier performance can be achieved through effective feature selection techniques and the optimization of hyperparameter values.
Aim: Performance measures of multiple machine learning classifiers are improved by fine tuning its hyperparameters using various type of meta-heuristic algorithms including swarm intelligent, physics, math, and bio-based, etc. Their performance measures are compared to find the best meta-heuristic algorithm in the context of code smell detection and its impact is evaluated based on statistical tests.
Method: This study employs sixteen contemporary and robust meta-heuristic algorithms to optimize the hyperparameters of two machine learning algorithms: Support Vector Machine (SVM) and k-Nearest Neighbors (k-NN). The No Free Lunch theorem underscores that the success of an optimization algorithm in one application may not necessarily extend to others. Consequently, a rigorous comparative analysis of these algorithms is undertaken to identify the best-fit solutions for code smell detection. A diverse range of optimization algorithms, encompassing Arithmetic, Jellyfish Search, Flow Direction, Student Psychology Based, Pathfinder, Sine Cosine, Jaya, Crow Search, Dragonfly, Krill Herd, Multi-Verse, Symbiotic Organisms Search, Flower Pollination, Teaching Learning Based, Gravitational Search, and Biogeography-Based Optimization, have been implemented.
Results: In the case of optimized SVM, the highest attained accuracy, AUC, and F-measure values are 98.75%, 100%, and 98.57%, respectively. Remarkably, significant increases in accuracy and AUC, reaching 32.22% and 45.11% respectively, are observed. For k-NN, the best accuracy, AUC, and F-measure values are all perfect at 100%, with noteworthy hikes in accuracy and ROC-AUC values, amounting to 43.89% and 40.83%, respectively.
Conclusion: Optimized SVM exhibits exceptional performance with the Sine Cosine Optimization algorithm, while k-NN attains its peak performance with the Flower Optimization algorithm. Statistical analysis underscores the substantial impact of employing meta-heuristic algorithms for optimizing machine learning classifiers, enhancing their performance significantly. Optimized SVM excels in detecting the God Class, while optimized k-NN is particularly effective in identifying the Data Class. This innovative

Submitted: 28 Nov. 2023; Revised: 4 Mar. 2024; Accepted: 3 Apr. 2024; Available online: 9 Apr. 2024

fusion automates the tuning process and elevates classifier performance, simultaneously addressing multiple longstanding challenges.

Keywords: Code Smell, Machine Learning, Meta-heuristics, Support Vector Machine, k-Nearest Neighbors, Optimization

1. Introduction

Using software is an integral part of our lives. They are embedded in every aspect of our existence, like education, transportation, entertainment, communication, healthcare, and security. Software systems have become complex and colossal with advancements in science and technology [1]. Designing and developing them takes a mere 30–40% of effort in the complete life cycle of software; the rest is dedicated to maintaining them [2]. Maintaining it includes continuously adding, deleting, and changing artefact functionalities to meet users' needs and satisfaction, which requires more resources and effort [3]. The un-involvement of the maintenance team in the development phase, time crunch, implementation of substandard design practices, tight deadlines, and inexperience of developers provides bedding ground for the spread of code smells throughout the system [4, 5]. A code smell is not a syntax error but may lead to it. It is a structural flaw that violates fundamental design principles and deteriorates code quality [6]. Furthermore, it makes code more complicated to understand and maintain and prevents code from changing, contributing to technical debts. So, it is best to identify and eradicate them whenever a new feature is added, while fixing a bug or during code reviews. It can be corrected by small and disciplined changes in code called refactoring. It is restructuring internal design but ensuring no change in external behaviour. Refactoring improves code quality and reverses software entropy. It makes the system more readable, understandable, efficient, flexible, and maintainable. Identifying and detecting smells is the first step in refactoring, making the system more robust and contemporary.

Code smell detection has several real-world applications in software development and maintenance. Smell detection helps developers identify code areas that may benefit from refactoring. By addressing code smells, developers can enhance code maintainability and readability, reducing technical debt and making the code base more sustainable. Incorporating smell detection as part of the quality assurance process ensures that newly developed or modified code adheres to best practices. In legacy systems where code has accumulated over time, identifying and mitigating code smells can be crucial for improving the health of the code base. It is essential when introducing new features, fixing bugs, or integrating modern technologies. Integrating code smell detection into Continuous Integration (CI) and Continuous Deployment (CD) pipelines ensures that any new changes introduced to the code base adhere to coding standards and best practices. Certain code smells are indicative of potential sources of bugs or errors. By proactively addressing these smells, developers can reduce the likelihood of introducing bugs and enhance the overall reliability of the software. Some code smells, such as duplicated code or inefficient algorithms, can impact the performance of the software. Detecting and addressing these smells can lead to performance improvements in the application. Code smell detection tools can be integrated into various development environments and IDEs, making it convenient for developers to identify and address issues during the coding process. In summary, code smell detection is a valuable practice with tangible benefits regarding code quality, maintainability, and team

collaboration. It contributes to the overall improvement of software development processes and the longevity of software systems.

Various code smell detection tools based on the visualization [7], machine based approach [8], and metric evaluation [9] are available in the market and they are of manual [10], automatic [11], and semi-automatic [12] in nature. Although code smell detection tools function effectively, subsequent research has shown essential flaws that jeopardize their widespread use. The agreement between different detectors is also impaired. Tools' strategies rely heavily on setting up detection rules and threshold values. Engineers need in-depth technical knowledge of code smells in order to define these rules [13]. Another problem is that code smells picked up by current detectors can be interpreted differently by specialists. More crucially, to identify smelly code components from non-smelly ones, the majority of smells require the specific threshold values, and naturally, the choice of threshold significantly impacts their count. Additionally, full consideration of size, domain, design, and complexity is typically lacking, which casts doubt on the veracity of other performance indicators [14]. The usage of code smell rules, using insufficient information, and metrics threshold levels are all overcome by supervised machine learning approach.

A supervised machine learning algorithm feeds in independent variables, commonly called training data, to determine the dependent variable's value and improves by learning through examples [15]. Performance measures are assessed, and the algorithm improves response from the difference between expected and generated output. Techniques like hyperparameter tuning [16], SMOTE [17], feature engineering [18], feature selection [19], etc., can be used to enhance results further. When using a machine learning approach, establishing rules and setting thresholds is left up to the algorithm rather than experts, significantly reducing time and effort [20].

This study uses two supervised machine learning classifiers, Support Vector Machine and k-Nearest Neighbors, to identify smelly instances. These classifiers employ a set of hyperparameters to enhance their results, and by choosing the appropriate values of hyperparameters, one may minimize error [21]. A hyperparameter is an external configuration to the model whose value must be defined by an expert as it cannot be determined from the data. The grid search technique can also improve the performance of machine learning algorithms [22]. However, it has many disadvantages, and for an algorithm to work successfully, a specialist must choose hyperparameter values. It takes specialized knowledge, intuition, and frequent trial and error for the best outcomes. It becomes impossible when the number of hyperparameters grows as evaluations grow exponentially. Therefore, meta-heuristic algorithms are employed to choose the appropriate values for the hyperparameters of machine learning algorithms to overcome this challenge and do away with the requirement for experts [23].

Meta-heuristic algorithms are high-level, problem-independent techniques that use gradient-free mechanisms and provide near-optimal solutions to highly complex real-world problems within limited computing time [24]. They search for a solution(s) in a search space that minimizes or maximizes an objective function while fulfilling certain constraints. The success of a meta-heuristic algorithm depends on two processes: exploration and exploitation. Diversification ensures that the whole search space is explored and not confined to specific areas, whereas, in intensification, certain better regions are explored more thoroughly to find a better solution. We have employed various meta-heuristic algorithms, which are stochastic in nature, exploring the search space and exploiting it for the best solution [25].

The no Free Lunch theorem states that no single optimization technique can solve all optimization problems [26]. This theorem underscores that the efficacy of an algorithm for

one application may not necessarily translate to success in another optimization problem. It led to the development of more than three hundred meta-heuristic algorithms for conquering numerous optimization problems. Consequently, the prudent approach involves implementing and comparing optimization algorithms to identify the most apt solution for a given context. Their categorization will guide us in understanding their basic work principles and strategies. They are categorized as follows:

- **Evolutionary:** These algorithms are inspired by Darwin's theory of survival of the fittest. The iterative selection, crossover, and mutation process make the stochastically generated population fitter. Evolutionary Programming [27], Genetic Algorithms [28], and Differential Evolution [29] are some evolutionary algorithms.
- **Swarm:** These algorithms utilize the social behaviour and hunting strategies of the genus of animals. Animals or insects work together in an organized manner and constantly interact to explore the entire search space and converge when necessary [30]. Examples of swarm-based algorithms are Particle Swarm Optimization [31], Ant Colony Optimization [32], etc.
- **Physics:** These algorithms imitate physical principles of the universe, such as gravitation, kinematics, fluid mechanics, and electromagnetism [33]. They can be categorized into thermodynamics, classical mechanics, optics, etc. Some physics-based algorithms are Multi-Verse Optimizer [34], Nuclear Reaction Optimization [35], etc.
- **Human:** These algorithms are inspired by the characteristics and behaviour of the human population. Brain Storm Optimization [36] and Battle Royale Optimization [37] are some examples.
- **Others:** Bio-inspired algorithms are based on interactions or biological processes observed in nature. Examples are Virus Colony Search [38], Earthworm Optimization [39], etc. Math-based algorithms such as Hill Climbing always move towards the peak to aim for a better solution [40]. Moreover, the Sine Cosine Algorithm explores and exploits search space using a mathematical model based on sine and cosine functions [41].

1.1. Motivation

Code smell detection has long been a focal point in software engineering research. This study pioneers a transformative approach by integrating meta-heuristic algorithms to amplify the performance of machine learning classifiers, offering a groundbreaking solution to enduring challenges. Employing an optimization algorithm eliminates the need for an expert and the painful task of finding the best hyperparameter values, automating and simplifying the whole process. This innovative fusion elevates classifier performance and presents a profound breakthrough in the field, addressing multiple long-standing issues concurrently.

The following research underscores the influence of meta-heuristic algorithms to optimize machine learning classifiers for code smell detection. The research delves into a comprehensive comparison of sixteen distinct meta-heuristic techniques, evaluating their efficacy in identifying and addressing smells within source code. In this research, the focus lies on the utilization of optimization algorithms to obtain optimal hyperparameter values for SVM and k-NN. A diverse range of optimization algorithms, encompassing Arithmetic, Jellyfish Search, Flow Direction, Student Psychology Based, Pathfinder, Sine Cosine, Jaya, Crow Search, Dragonfly, Krill Herd, Multi-Verse, Symbiotic Organisms Search, Flower Pollination, Teaching Learning Based, Gravitational Search, and Biogeography-Based Optimization, have been implemented.

A comprehensive comparative analysis examines the performance of machine learning classifiers across three scenarios: absence of optimization, grid search application, and optimization implementation. Key performance metrics, including Accuracy, ROC Area Under the Curve (ROC-AUC), F-measure, and execution time, are meticulously documented for analytical purposes. The techniques are implemented 25 times, Acknowledging the stochastic nature of meta-heuristic algorithms. The resultant best and average values are considered for evaluation. Furthermore, a juxtaposition is drawn between these novel meta-heuristic methods and classical algorithms, such as Differential Evolution, Particle Swarm Optimization, Genetic Algorithm, and Simulated Annealing, enhancing the breadth and depth of the comparative study.

The overall contribution of this paper is:

- Enhancing the performance of machine learning classifiers through the utilization of diverse meta-heuristic algorithms.
- Demonstrating the profound influence of meta-heuristic algorithms in optimizing machine learning classifiers, specifically in the context of code smell detection.
- Identifying the optimal meta-heuristic technique for effectively detecting code smells within source code.
- Disclosing the most readily detectable code smell instances.
- Establishing a foundational reference study for prospective qualitative and quantitative comparative research across various domains.

The research paper is structured across seven distinct sections. Commencing with an introductory segment, the paper outlines fundamental concepts, underscores the study's necessity, and articulates its contributions. The introduction is followed by an overview of related research and the pivotal role of the current study. The third section comprehensively details the experiment setup and methodology, accompanied by an illustrative workflow. Results, their in-depth analysis, and statistical tests comprise the fourth section, followed by an expansive discussion in the fifth. The sixth section meticulously examines potential threats to validity, providing corresponding mitigation strategies. The paper culminates in a conclusive seventh section, encapsulating final thoughts and avenues for future exploration. Additionally, the machine learning algorithms employed and concise profiles of the sixteen meta-heuristic algorithms employed are expounded upon within the appendices.

2. Related work

The use of machine learning and optimization algorithms represent highly sought-after and crucial areas of research. Extensive investigations have been undertaken to enhance the efficiency of machine learning algorithms employing diverse techniques, among which the utilization of meta-heuristic algorithms holds significance. Optimization algorithms serve a dual purpose within this landscape like facilitating feature selection and hyperparameter tuning for machine learning algorithms. Moreover, these algorithms find utility in establishing detection rules for code smells, employing tailored threshold values and metrics. Applying optimization algorithms extends to prioritizing refactoring for multiple code smells, predicated on factors such as severity and risk in the context of extensive software systems. The following section outlines pertinent research efforts in this domain.

Hassaine et al. utilized a machine learning-inspired technique called an Immune-based Detection Strategy that imitated the immune system of the human body [42]. IDS is based on the Artificial Immune Systems (AIS) algorithm, which mimics the defense mechanisms

of the human immune system. The authors drew a parallel between the human body and system design to develop a detection method that identifies smelly classes, equivalent to pathogens, using some features of the classes in the form of metrics. Compared to DECOR [43] and Bayesian Belief Networks, IDS outperformed in precision and computation time.

Maiga et al. introduced SMURF – Support Vector Machines that consider practitioners' feedback [44]. SMURF was compared with DETEX [43] and BDTEX [45]; it performed better in accuracy, precision, and recall. Fontana et al. implemented multiple variations and boosted versions of J48, Random Forest, Naive Bayes, JRip, SMO, and SVM, constituting 32 machine-learning algorithms to detect four code smells. They concluded that J48 and Random Forest yield the highest performance, and support vector machines are the worst. Boosting only sometimes helps; in some cases, it diminishes performance [46].

Kessentini et al. proposed a multi-objective genetic programming algorithm (MOGP) to generate rules for automatically detecting code smells in Android applications. They identified detection rules for ten smells and evaluated their technique on 184 Android projects. Results projected that average correctness was more than 82% and an average relevance of 77% based on the feedback of active developers of mobile apps [47]. Kaur et al. designed a new meta-heuristic optimization algorithm inspired by sandpipers' searching and attacking behaviours, known as the Sandpiper Optimization Algorithm (SPOA). They collaborated SPOA with B-J48 pruned machine-learning approach to detect five code smells in three open-source software [48].

Jain et al. applied three hybrid feature selection techniques with ensemble machine learning algorithms to improve the performance in detecting code smells. Seven machine learning classifiers with different kernel variations, along with three boosting designs, two stacking methods, and bagging, were implemented. Combining filter-wrapper, filter-embedded, and wrapper-embedded methods was executed for feature selection. After application of hybrid feature selection, performance measure increased, accuracy by 21.43%, ROC AUC value by 53.24%, and F-measure by 76.06% [16].

In other work, Jain et al. implemented 32 machine learning algorithms with feature selection that drastically eliminated the dimensionality curse and improved performance measures. Two correlation methodologies, brute force and random forest, were used to discard irrelevant features with three filter methods: mutual information, fisher score and univariate ROC-AUC. Results showed that the accuracy of machine learning models had surged up to 26.5%, F-measure by 70.9%, the area under the ROC curve had levelled up to 26.74%, and average training time has reduced up to 62 secs as compared to measures of models without feature selection [49].

Boussaa et al. proposed a promising technique to identify detection rules for code smell detection. Two populations evolved simultaneously. The first produced a set of detection rules for detecting code smells, and the second introduced artificial code smells to support the main objective of the first population. When tested on four open-source Java systems, this technique outperformed two single population-based meta-heuristics, Genetic Programming and Artificial Immune Systems [50].

Similarly, Kessentini et al. parallelly used genetic programming to generate code smell detection rules and genetic algorithms to produce code smell examples. Cooperative P-EA outperforms single population evolution and random search [51]. Sahin et al. implied code smell detection as a bilevel problem [52]. They used genetic programming for the upper-level problems, i.e., detection rules and generated artificial code smells for lower-level problems. However, there was no parallelism in this bilevel approach; levels were executed

serially. This technique outperformed Genetic Programming, Competitive Coevolutionary Search [50] and non-search-based methods.

Mansoor et al. used multi-objective genetic programming (MOGP) to find the most optimized detection rules to maximize the detection of smells and minimize false detection problems. Five code smells were inspected on seven large open-source systems, and the algorithm achieved 87% precision and 92% recall [53]. Saranya et al. proposed Euclidean distance-based Genetic Algorithm and Particle Swarm Optimization (EGAPSO) to develop detection rules that outperformed other detection methods like Genetic Algorithm, DECOR, Parallel Evolutionary Algorithm, and Multi-Objective Genetic Programming. The approach was tested on open-source projects like the Gantt Project and Log4j to identify the five code smells [54].

Kannan developed hybrid particle swarm optimization with mutation (HPSOM) to formulate detection rules using appropriate metrics and thresholds. He then compared its performance with other evolutionary techniques like the parallel evolutionary algorithm, genetic algorithm, genetic programming, and particle swarm optimization. HPSOM outperformed all of them by achieving a precision of 94% and recall of 92%. He worked with nine open-source projects and detected five code smells: blob, data class, spaghetti code, functional decomposition, and feature envy [55]. Moatasem et al. used a whale optimization algorithm to formulate ideal detection rules for nine code smells. Equations were tested on five medium and large-size open-source projects. Results were better than other search-based algorithms; 94.24% precision and 93.4% recall were observed [56].

Amal et al. evaluated a refactoring series to make the system more robust using a genetic algorithm and artificial neural network (ANN) [57]. They compared their techniques with other search-based refactoring techniques, such as the IGA technique presented by Ghannem et al. [58] and a design defect detection and correction tool called JDeodorant [59]. Dea et al. used distributed evolutionary algorithms where many evolutionary algorithms with different adaptations (fitness functions, solution representation, and change operators) are implemented in parallel to get a series of refactoring. Cooperative D-EA outperforms single population evolution and random search based on a benchmark of eight sizable open-source systems where more than 86% of code smells are fixed using the suggested refactoring [60].

Saranya et al. used the Strength Pareto Evolutionary Algorithm (SPEA) to prioritize the list of refactorings. Blob, Functional Decomposition, Shotgun Surgery, Data Class, Schizophrenic Class, and Swiss Army Knife were considered and tested on two open-source systems, Xerces-J and J Hot Draw. SPEA outperformed Chemical Reaction Optimization (CRO) and Non-dominated Sorting Genetic Algorithm in prioritizing code smell correction tasks [61].

Large-scale systems have a volume of code smells, and prioritizing them according to risk, impact, importance, and severity is an efficient way to eliminate them. Ouni et al. used chemical reaction optimization to find a series of refactoring to remove smells according to the risk and other factors involved. Seven code smells were tested on five medium to large-scale open-source systems. The proposed technique outsmarted existing methods compared to Genetic Algorithm, Simulated Annealing, and Particle Swarm Optimization [62]. Using the Sandpiper Optimization Algorithm, Kaur et al. detected the severity of five harmful code smells, namely blob, feature envy, data class, functional decomposition, and spaghetti code. The approach was tested on four open-source Java software: Gantt-Project, Log4j, and two different versions of Xerces. Studies showed that many code smells could be refactored with a severe decrease in refactoring effort [63].

This work utilizes the influence of sixteen powerful meta-heuristic algorithms to optimize machine learning algorithms. This approach addresses problems such as evaluating the

best hyperparameter values of machine learning algorithms to elevate their performance. This fusion eliminates needing an expert, reduces time and effort, and effectively deciphers complex software engineering challenges.

3. Research methodology

The following section delves into the research questions, studies answers, description of code smells analyzed, datasets used, and complete experimentation settings.

3.1. Research questions addressed

This study aims to investigate the following research questions:
- **RQ 1:** Does using meta-heuristic algorithms for optimizing machine learning classifiers boost their performance for detecting code smell in complex software systems?
- **RQ 2:** How significant is the impact of optimization of machine learning algorithms with meta-heuristic techniques on its overall performance?
- **RQ 3:** Given the meta-heuristic algorithms, which yields the best performance in optimizing classifiers to detect code smell and why?
- **RQ 4:** How does our approach perform compared to existing machine learning based techniques?

3.2. Code Smells investigated

This study entails the optimization of two distinct machine learning classifiers by utilizing a comprehensive array of sixteen selected meta-heuristic algorithms, a strategy aimed at refining performance metrics. The primary focus of this optimization effort is detecting four distinct types of code smells, each of which bears distinctive characteristics and implications. Specifically, two class-level code smells [64] under scrutiny are the Data Class and the God Class. Data Class is a passive container for data, housing attributes, getters, and setters intended for use by other encapsulating classes. This class does not engage in the execution of substantial operations on its stored data. It impacts data abstraction and encapsulation properties of the system. It can be refactored with the Encapsulate Collection, Move Method, Extract Method, Encapsulate Field, and Hide Method.

God Class is characterized by its tendency for extensive functionality implementation, leveraging attributes sourced from various other classes. This behaviour results in a notably intricate and expansive class structure that is difficult to understand and maintain. It promotes code duplication and complex methods. It affects cohesion, coupling, complexity, and size of the system. It can be refactored with Extract Class, Extract Subclass, Extract Interface, and Duplicate Observed Data. Further delving into the method-level code smells [65], two distinct categories are investigated: Feature Envy and Long Method. Feature Envy manifests when a method tends to access attributes originating from external classes while interacting with data derived from these classes. It harms the coupling and data abstraction properties of code. It can be treated with Extract Method and Move Method refactoring.

The long method is overly extensive and draws information from other methods. These methods often seek to centralize a class's intelligence and encompass many features. It impacts coupling, cohesion complexity, and the size of the whole system. One can eliminate

the Long Method with the Extract Method, Replace Temp with Query, Introduce Parameter Object, Preserve Whole Object, Replace Method with Method Object, and Decompose Conditional refactoring techniques. While individually diverse in their manifestations, these code smells collectively embody some of the most insidious and prevalent issues encountered within software code bases [66]. Their systematic detection and subsequent remediation are pivotal to enhancing software quality, maintainability, and comprehensibility.

3.3. Datasets used

In this research, we have used four datasets curated by Fontana et al. [46] to facilitate the classification of specific code smells. These datasets have been assembled from 74 compilable Java systems sourced from the Qualitas Corpus [67]. Collectively, these systems span a diverse spectrum of application domains and exhibit a wide range of sizes, thereby endowing our research with a robust and comprehensive foundation. Datasets included an intra-system setup to prevent machine learning models from succumbing to over-fitting. List of all heterogeneous systems is included in the Table 1 of supplementary file[1]. They developed the Design Features and Metrics for Java (DFMCFJ) tool, underpinned by the Eclipse JDT Library, which extracts a rich array of object-oriented metrics at multiple granularities, spanning project, package, class, and method levels.

Each dataset comprises 420 data points, partitioned into 280 negative samples, signifying the absence of code smell, and 140 positive samples, denoting presence. Datasets are rooted in a comprehensive assessment of object-oriented metrics, spanning multiple strata of code design, such as coupling, complexity, cohesion, and size. Details of all metrics used in datasets are mentioned in the Table 2 of supplementary file. The datasets are judiciously leveraged by stratified sampling techniques, thus generating balanced and labeled datasets. It is imperative to note that each entry within these datasets is expressly associated with either a method or a class. Each row is labeled with the help of Advisor (Code smell detection tools such as PMD, iPlasma, Fluid Tool, and Antipattern Scanner) and validated by trained MSc students after thorough discussion.

The metrics encompass a comprehensive view of code design focused on method-level code smells, extending across project, package, class, and method levels. The method-level datasets harnessed 82 distinct metrics. Conversely, the datasets dedicated to class-level code smells have a set of metrics spanning project, package, and class levels, totaling 61 in number. This approach ensures that our research is firmly grounded in a wealth of empirical data, encompassing a multifaceted view of code characteristics. Thus, it is poised to yield comprehensive insights into code smell detection. Datasets are made available by Fontana et al.[2].

3.4. Experimentation setup

This research aims to enhance the performance of two prominent machine learning classifiers: Support Vector Machine (SVM) and k-Nearest Neighbors (k-NN). This enhancement is pursued by utilizing sixteen distinct meta-heuristic algorithms, expertly calibrated to fine-tune the hyperparameters governing these classifiers. A comprehensive assessment and comparative analysis of the efficacy and impact of these meta-heuristic algorithms within

[1]Details of Datasets – https://drive.google.com/file/d/1Jt3jnRDUKgCvN-ZUM6xwtZTut8GuIFcL/view?usp=sharing

[2]Datasets – https://drive.google.com/file/d/15aXc_el-nx4tQwU3khunQ-I5ObSA1-Zb/

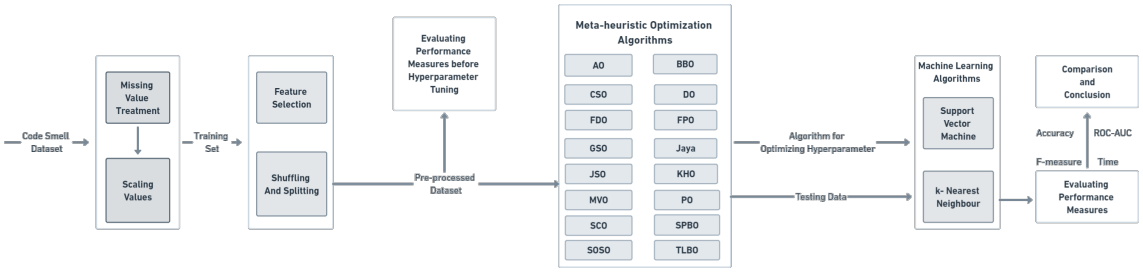


Figure 1. Workflow

machine learning is pursued. This research unfolds within the computational domain of Python [68], with the scikit-learn [69] framework serving as the foundational infrastructure. To visually represent the holistic research process, we have encapsulated the workflow of our study in Figure 1.

The data pre-processing ensures the cleanliness and readiness of datasets before they are divided into training and test sets [70]. This preliminary data grooming is essential, as it directly influences the quality and appropriateness of the data employed for model training. It ameliorates model performance, reduces training duration, mitigates over-fitting risks, and enhances model interpretability [71]. The following steps are taken to prepare datasets. The missing data values are replaced with a zero value due to intra-system settings. A scaling operation establishes an equitable ground for our independent variables. This normalization strategy serves the dual purpose of bridging any inherent gaps between features and curbing the potential introduction of bias.

The next step in the data refinement entails identifying and eliminating constant, quasi-constant, and duplicated features. These categories encapsulate features that either exhibit an unchanging value across instances (constant features), furnish redundant or repetitive information (duplicate features), or verge on maintaining nearly identical values for every instance (quasi-constant features) [72]. The independent variables should not be correlated and reasonably correlate with the dependent variable. Thus, correlated independent variables are precisely identified and eliminated. Pearson's correlation coefficient [73] is employed for the same, a well-established statistical metric renowned for its adeptness in quantifying the linear relationship between two variables.

The dataset is randomly partitioned into two sets for training and testing purposes. The training dataset is formulated, constituting 80% of the entire dataset, while the test dataset accounts for the remaining 20%. K-fold cross-validation is employed to assess a predictive model's performance and generalization ability. It scrutinizes the efficacy and proficiency of machine learning models when confronted with previously unseen data. This method divides the dataset into K parcels of identical size, denoted as "folds". We set the value of K to 10, signifying ten equivalent folds [74]. The model is iteratively trained on $k-1$ folds while reserving one fold for validation. This cyclic process iterates K times, each fold having a turn as the validation set. The final model's performance is the aggregation of performance scores garnered across all K iterations. Machine learning algorithms are implemented, and their hyperparameters are obtained from meta-heuristic optimization techniques. Employing meta-heuristic algorithms explores and scrutinizes the parameter space to identify the optimal hyperparameters that give the peak performance of the machine learning classifier. The working of meta-heuristic algorithms is presented in Figure 2, which is explained in the following section:

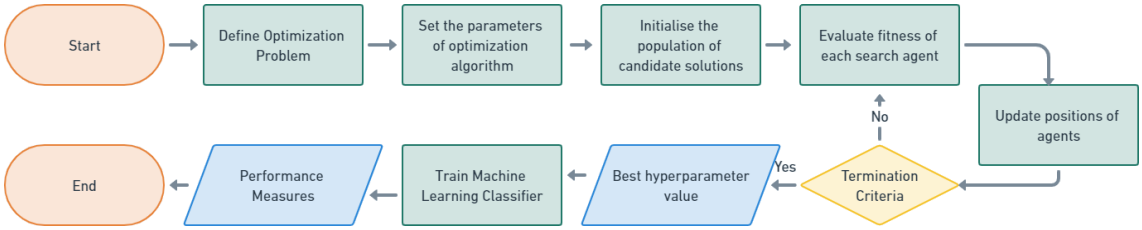


Figure 2. Flowchart: Steps involved in working of meta-heuristic algorithms

1. **Define the optimization problem.** At its core, the optimization task revolves around minimizing errors in machine-learning classifiers. This entails identifying optimal hyperparameter values, a prerequisite before the classifier's training phase, and directing and shaping its behaviour. The Support Vector Machine (SVM) has two critical hyperparameters, "C" and "gamma," for optimization. Simultaneously, the k-Nearest Neighbors (k-NN) classifier undergoes refinement by selecting optimal "n_neighbors" and "p" values. In the case of SVM, the Radial Basis Function (RBF) kernel is a natural choice, endowed with a track record of superior performance.
2. **Set parameter values for meta-heuristic algorithm.** The parameter values, such as population size, iteration count, generation specifications, etc., are initialized. These selected parameter values for meta-heuristic algorithms and the hyperparameter spectrum of machine learning classifiers are thoughtfully detailed in Table 1. Arriving

Table 1. Parameter values used for meta-heuristics algorithms and range of hyperparameters for machine learning classifiers

Ref	Meta-heuristic algorithms	Year	Category	Parameters
[75]	Arithmetic Optimization	2021	Math	size = 5, alpha = 5, mu = 0.5
[76]	Jellyfish Search Optimization	2021	Swarm	jellyfishes = 5, eta = 4, beta = 3, gamma = 0.1, c_0 = 0.5
[77]	Flow Direction Optimization	2021	Physics	size = 5, beta = 8
[78]	Student Psychology Based Optimization	2020	Human	size = 5, generations = 50
[79]	Pathfinder Optimization	2019	Swarm	size = 5, generations = 50
[80]	Sine Cosine Optimization	2016	Math	solutions = 5, a_linear_component = 2, r1 = 2
[81]	Jaya Optimization	2016	Swarm	size = 5, generations = 50
[82]	Crow Search Optimization	2016	Swarm	size = 5, ap = 0.02, fL = 0.02
[83]	Dragonfly Optimization	2016	Swarm	size = 3, generations = 50
[84]	Krill Herd Optimization	2016	Swarm	size = 5, generations = 50, mutation_rate = 0.1, eta = 1, c_t = 1, mu = 1, elite = 0
[85]	Multi-Verse Optimization	2015	Physics	universes = 5
[86]	Symbiotic Organisms Search	2014	Bio	size = 5, eta = 1, generations = 50, mutation_rate = 0.1
[87]	Flower Pollination Optimization	2012	Evolutionary	flowers = 3, gamma = 0.5, lamb = 1.4, p = 0.8, beta = 1.5
[88]	Teaching Learning Based Optimization	2012	Human	size = 5, generations = 50
[89]	Gravitational Search Optimization	2009	Physics	swarm_size = 5
[90]	Biogeography-Based Optimization	2008	Bio	size = 5, mutation_rate = 0.1, elite = 0, eta = 1, gens = 50
[91]	Support Vector Machine	–	–	C = [10, 1000], gamma = [0.05, 10], kernel = rbf
[92]	k-Nearest Neighbors	–	–	n_neighbors = [3, 50], p = [1, 2]

at these optimal values is underpinned by an exhaustive investigative process involving comprehensive research, literature survey, and methodical empirical experimentation. These chosen values are supported by foundational research. Furthermore, the machine learning algorithm's hyperparameter range is selected by drawing insights from research, experimentation, and practical experience.

3. **Generate initial population.** During this phase, the algorithm initializes the population of the candidate solution, a process that can involve randomized generation or employ alternative strategies [93]. For optimization, a comprehensive ensemble of sixteen meta-heuristic optimization algorithms is harnessed. These encompass Arithmetic, Jellyfish Search, Flow Direction, Student Psychology Based, Pathfinder, Sine Cosine, Jaya, Crow Search, Dragonfly, Krill Herd, Multi-Verse, Symbiotic Organisms Search, Flower Pollination, Teaching Learning Based, Gravitational Search, and Biogeography-Based Optimization. The objective is determining the optimal hyperparameter values for classifiers to augment the performance. The behaviour, working principle, and learning equation of each meta-heuristic algorithm are mentioned in the Appendix B.
4. **Fitness evaluation.** In this phase, the fitness of each candidate solution undergoes scrutiny. This undertaking entails training the machine learning classifier, which utilizes the hyperparameters intrinsic to each candidate solution. Subsequently, their performance is appraised primarily on a validation dataset. A designated performance metric, accuracy or F-measure, is wielded as the discerning fitness function, serving as the threshold to quantify the efficacy of each solution.
5. **Updating Position Vectors.** The candidate solutions are updated based on the information obtained in the fitness evaluation step and the rules of the meta-heuristic algorithm. This involves adjusting the position of each search agent, updating the best-performing agent, or selecting new agents to replace under-performing ones.
6. **Termination Criteria.** The position updation continues until a stopping criterion is met. This could involve checking if the maximum number of iterations has been reached, if the best-performing solution has not improved in a certain number of iterations, or if the solutions have converged to a certain level of accuracy. For this study, stopping criteria are set to 50 iterations because experimentation found that this number is sufficient to converge to an appropriate solution. As optimization algorithms are stochastic, the process is repeated 25 times to achieve the best and average values.
7. **Output.** If the termination criteria have been met, the algorithm finds the best-performing hyperparameters found; otherwise, it returns to step 4 and continues the optimization process.
8. **Training.** In this step, the machine learning classifier is trained with the best hyperparameter values derived from the above step. Our study has chosen SVM and k-NN for optimization, and its working is mentioned in the Appendix A.
9. **Evaluation.** The performance measures of machine learning classifiers, such as accuracy, F-measure, and ROC-AUC, are evaluated for analysis and comparison purposes.

By using meta-heuristic algorithms to search for the best hyperparameter values, we can avoid the time-consuming and error-prone process of manual tuning [94]. The aim is to minimize the error component, that is, the difference between the predicted and actual value of the target variable. Accuracy, ROC Area Under the Curve (ROC-AUC), F-measure, and execution time are recorded for analysis after optimization. Deviation from the standard and time for one iteration is an average of 25 executions. Further, the final performance is compared with the scenario when no such optimization technique is used, the grid search method is used, and classic meta-heuristic algorithms such as Differential

Evolution, Particle Swarm Optimization, Genetic Algorithm, and Simulated Annealing are used. Results are presented in tabular form in the next section with a qualitative and quantitative analysis.

3.5. Performance measures

Following are the performance measures for classification problems that have been used to analyze and compare machine learning algorithms:

3.5.1. Accuracy

It is the sum of all correctly predicted code smells divided by the total number of smells present in the code. It can be calculated by the formula:

$$\frac{Detected\ Code\ Smells}{Total\ Code\ Smells\ Present}$$

3.5.2. F-measure

F-measure is the weighted harmonic mean of recall and precision.

$$F\text{-measure} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Precision is the number of smells predicted as smelly and are also actually smelly. It is calculated as:

$$\frac{(Present\ Code\ Smells) \bigcap (Detected\ Code\ Smells)}{(Detected\ Code\ Smells)}$$

Recall is the number of instances that are actually smelly and are also predicted correctly as smelly. It is calculated as:

$$\frac{(Present\ Code\ Smells) \bigcap (Detected\ Code\ Smells)}{(Present\ Code\ Smells)}$$

3.5.3. ROC-AUC

Receiver Operating Characteristics (ROC) is a plot of the False Positive Rate (on the x-axis) versus the True Positive Rate (on the y-axis) for every possible classification threshold. The area calculated under the ROC curve is known as ROC-AUC.

$$\text{ROC-AUC} = \frac{1 + TP_{\text{rate}} - FP_{\text{rate}}}{2}$$

It represents the probability that a machine learning model ranks a randomly chosen positive observation higher than a randomly chosen negative observation, and thus it is a useful metric even for skewed datasets [95].

4. Results and analysis

Leveraging meta-heuristic algorithms to attain the optimal hyperparameter values for machine learning algorithms is a prominent strategy for achieving peak performance. In this study, the Support Vector Machine and k-Nearest Neighbors undergo optimization by using sixteen meta-heuristic methods. The following evaluation encompasses a comprehensive comparison and scrutiny of their respective performance measures, facilitating the identification of the most effective optimization algorithm. This analysis accounts for four distinct code smells, categorized into class-level and method-level varieties. The research findings present results via tables and in-depth analytical modules.

Tables 3 through 6 comprehensively examine the performance measures for the Support Vector Machine (SVM) in conjunction with various meta-heuristic algorithms across all four distinct code smells. Tables 8 to 11 meticulously present the performance metrics of k-Nearest Neighbors (k-NN) when optimized with various meta-heuristic algorithms, encompassing all four distinct code smells. Notably, all sixteen meta-heuristic algorithms are executed twenty-five times to derive the average values for all performance measures. Across these twenty-five iterations, the most exceptional performance measure is recorded for each code smell. This measure is compared against the original performance metrics obtained when no optimization technique is applied. This comparative analysis seeks to elucidate the precise impact that optimization algorithms wield over machine learning processes.

It is important to note that in cases where the F-measure is not always measurable, the difference related to this metric is omitted from consideration. Additionally, standard deviation values are computed, providing insights into the degree of variation within the data. Furthermore, the time required per iteration is documented, offering a glimpse into the computational efficiency of these optimization algorithms. As part of this comprehensive evaluation, the performance of the selected meta-heuristic algorithms is juxtaposed with that of four widely recognized and fundamental techniques: Genetic Algorithm, Differential Evolution, Particle Swarm Optimization, and Simulated Annealing. The best performance measures are denoted in bold to highlight the most outstanding results. Subsequently, this narrative will delve into a detailed and systematic analysis of how optimization impacts the performance of classifiers in the context of each specific code smell.

4.1. Support Vector Machine

Table 2 presents an extensive evaluation of the performance metrics, including accuracy, ROC-AUC, and F-measure, of the Support Vector Machine (SVM). This evaluation encompasses instances without optimization and when grid search is executed. In grid search, range of hyperparameter selected is as follows – C: [0.1, 1, 10, 100, 1000], gamma: [1, 0.1, 0.01, 0.001, 0.0001]. Without optimization, the God Class exhibits the highest values, recording accuracy and ROC-AUC of 73.89% and 84.88%, respectively. When employing grid search, best metrics are noted as 75% for accuracy and 73.69% for ROC-AUC. Notably, the Data Class demonstrates some gains in accuracy of 3.61%. With grid search ROC-AUC value always decreased, even up to 11.19%. It's noteworthy that grid search fails to improve results in all cases, even decreased in some, underscoring the necessity for alternative techniques to achieve optimal outcomes.

Table 2. Performance measures of Support Vector Machine (SVM)

	Without optimization			Grid search		
Code smells	Accuracy	ROC-AUC	F-measure	Accuracy	ROC-AUC	F-measure
Data class	67.92	62.78	0	71.53	55.83	**21.67**
Feature envy	64.17	68.72	**18**	61.94	67.25	0
God class	**73.89**	**84.88**	0	**75**	**73.69**	0
Long method	64.17	66.22	**18**	64.31	61.92	13

4.1.1. Data class

The Data Class detection outcomes are carefully presented in Table 3. Among the meta-heuristic algorithms, **Symbiotic Organisms Search Optimization** emerges as the unequivocal champion, consistently exhibiting the maximum, optimal average, and most substantial improvements in all three performance metrics. Symbiotic Organisms Search Optimization attains a peak accuracy of 97.64% and the finest average accuracy of 97.64% while achieving an impressive increase of 29.72% (Δ_1) when juxtaposed with non-optimized results. Furthermore, this algorithm achieves highest ROC-AUC of 100% and a remarkable 99.96% as the best average value, with a minimal deviation of 0.15; these achievements correspond to a notable surge of 37.22% (Δ_2) compared to the non-optimized baseline. For F-measure, both Jellyfish Search and Symbiotic Organisms Search Optimization emerge as front runners, securing the highest maximum and optimal average value of 96%. Among the algorithmic contenders, Dragonfly Optimization appears the swiftest, boasting an execution time of 6.97 seconds per iteration. In contrast, Pathfinder Optimization is the most time-consuming option for detecting Data Class.

4.1.2. Feature envy

Table 4 encapsulates the findings identifying Feature Envy through SVM and diverse meta-heuristic algorithms. Crow Search Optimization stands out with its highest recorded accuracy of 86.11%. In parallel, Dragonfly Optimization attains the highest ROC-AUC and F-measure, registering remarkable values of 99.33% and 94.29%, respectively. When considering the average performance metrics, Symbiotic Organisms Search Optimization emerges as the frontrunner, achieving the best average accuracy of 94.36% and the highest average F-measure, 92.37%. These achievements come with minimal deviations of 1.34 and 1.18, respectively. Pathfinder Optimization secures the best average ROC-AUC value, an impressive 98.57%. Furthermore, Dragonfly Optimization is characterized by the most substantial improvements in accuracy and ROC-AUC values, attaining increments of 32.08% (Δ_1) and 30.61% (Δ_2), respectively, compared to the non-optimized baseline. It is also the swiftest optimization algorithm, with an execution time of merely 7.09 seconds per iteration. In stark contrast, Pathfinder Optimization ranks as the slowest algorithm in execution time, with an enduring 242.03 seconds per iteration. The results underscore **Dragonfly Optimization** as the most proficient algorithm for Feature Envy detection when coupled with SVM.

4.1.3. God class

Table 5 furnishes the outcomes pertinent to identifying the God Class, showcasing the results obtained when utilizing SVM with diverse meta-heuristic algorithms. **Sine Cosine**

Table 3. Performance measures of SVM optimized with meta-heuristic algorithms for Data Class

Performance measures	Accuracy				ROC-AUC				F-measure			Time of one
Optimization Algorithms	Best [%]	Avg [%]	Std	Δ_1	Best [%]	Avg [%]	Std	Δ_2	Best [%]	Avg [%]	Std	iteration [s]
Arithmetic Optimization	91.67	75.88	7.76	23.75	95.83	90.33	2.84	33.05	83.90	16.87	24.77	15.31
Jellyfish Search Optimization	97.50	97.50	0.00	29.58	98.67	98.67	0.00	35.89	**96.00**	**96.00**	0.00	36.65
Flow Direction Optimization	90.28	90.28	0.00	22.36	94.00	94.00	0.00	31.22	88.06	88.06	0.00	203.12
Student Psychology Based Optimization	96.39	93.72	9.04	28.47	99.33	97.92	3.83	36.55	94.67	90.88	18.55	47.70
Pathfinder Optimization	90.56	90.51	0.22	22.64	97.83	97.83	0.00	35.05	84.71	84.71	0.00	214.55
Sine Cosine Optimization	94.17	94.17	0.00	26.25	98.89	98.89	0.00	36.11	88.00	88.00	0.00	32.08
Jaya Optimization	90.28	87.10	7.28	22.36	96.11	94.72	3.76	33.33	77.90	65.44	28.56	13.54
Crow Search Optimization	85.83	74.39	7.09	17.91	96.83	89.96	3.20	34.05	87.71	34.63	24.08	25.38
Dragonfly Optimization	95.42	93.95	3.96	27.50	99.17	98.44	0.59	36.39	95.14	88.18	11.68	**6.97**
Krill Herd Optimization	82.78	80.98	1.41	14.86	90.17	89.05	0.99	27.39	78.81	75.53	1.93	21.30
Multi-Verse Optimization	91.53	86.73	9.48	23.61	98.11	98.11	0.00	35.33	86.71	71.32	30.66	29.21
Symbiotic Organisms Search Optimization	**97.64**	**97.64**	0.00	**29.72**	**100.00**	**99.96**	0.15	**37.22**	**96.00**	**96.00**	0.00	85.12
Flower Pollination Optimization	95.42	89.70	10.25	27.50	97.50	97.01	0.97	34.72	91.33	72.40	34.80	8.14
Teaching Learning Based Optimization	96.39	96.39	0.00	28.47	99.17	99.17	0.00	36.39	91.33	91.33	0.00	51.06
Gravitational Search Optimization	95.28	76.20	8.46	27.36	99.44	98.24	1.63	36.66	91.24	32.44	32.99	20.07
Biogeography-Based Optimization	94.03	83.57	12.92	26.11	97.56	96.79	1.76	34.78	91.14	41.58	41.08	37.80
Differential Evolution	66.67	66.67	0.00	−1.25	67.64	60.72	8.49	4.86	0.00	0.00	0.00	14.03
Particle Swarm Optimization	63.06	63.06	0.00	−4.86	65.44	60.03	7.23	2.66	0.00	0.00	0.00	36.5154
Genetic Algorithm	65.42	59.74	1.61	−2.50	68.42	56.70	5.16	5.64	9.00	5.76	4.32	9.23011
Simulated Annealing	66.67	64.37	0.68	−1.25	58.00	54.36	1.53	−4.78	10.00	0.80	2.71	13.14

Table 4. Performance measures of SVM optimized with meta-heuristic algorithms for Feature Envy

Performance measures	Accuracy				ROC-AUC				F-measure			Time of one
Optimization Algorithms	Best [%]	Avg [%]	Std	Δ_1	Best [%]	Avg [%]	Std	Δ_2	Best [%]	Avg [%]	Std	iteration [s]
Arithmetic Optimization	79.58	78.74	0.96	15.41	82.86	71.12	4.25	14.14	13.33	11.73	4.33	14.77
Jellyfish Search Optimization	85.69	83.88	6.14	21.52	93.06	93.06	0.00	24.34	82.05	82.05	0.00	39.67
Flow Direction Optimization	85.83	84.90	0.41	21.66	97.67	97.67	0.00	28.95	67.00	65.86	0.57	228.73
Student Psychology Based Optimization	84.58	80.31	5.24	20.41	92.50	91.69	2.12	23.78	51.00	46.92	13.84	51.60
Pathfinder Optimization	88.19	88.19	0.00	24.02	98.57	**98.57**	0.00	29.85	58.33	58.33	0.00	242.03
Sine Cosine Optimization	84.44	84.44	0.00	20.27	98.33	98.33	0.00	29.61	55.67	55.67	0.00	16.10
Jaya Optimization	84.44	79.33	6.26	20.27	96.94	96.94	0.00	28.22	61.33	49.07	24.53	14.49
Crow Search Optimization	86.11	58.20	8.91	21.94	92.50	82.43	4.76	23.78	81.13	13.18	27.95	19.21
Dragonfly Optimization	**96.25**	88.27	10.85	**32.08**	**99.33**	97.00	1.96	**30.61**	**94.29**	82.00	29.10	**7.09**
Krill Herd Optimization	65.42	65.42	0.00	1.25	57.83	57.69	0.34	−10.89	0.00	0.00	0.00	29.36
Multi-Verse Optimization	91.67	86.42	10.50	27.50	97.00	97.00	0.00	28.28	88.95	46.26	44.44	14.31
Symbiotic Organisms Search Optimization	95.56	**94.36**	1.34	31.39	99.00	98.25	0.75	30.28	93.81	**92.37**	1.18	88.54
Flower Pollination Optimization	77.22	74.97	3.61	13.05	93.61	91.26	7.35	24.89	41.33	24.80	20.25	8.98
Teaching Learning Based Optimization	84.72	83.57	0.34	20.55	97.56	97.56	0.00	28.84	61.33	59.41	0.39	54.32
Gravitational Search Optimization	79.03	65.16	3.43	14.86	92.56	89.52	4.94	23.84	64.57	2.58	12.65	16.40
Biogeography-Based Optimization	84.31	73.36	6.32	20.14	95.50	91.32	6.78	26.78	60.67	26.11	28.73	27.83
Differential Evolution	61.94	61.94	0.00	−2.23	67.00	65.80	2.75	−1.72	0.00	0.00	0.00	16.26
Particle Swarm Optimization	71.67	71.67	0.00	7.50	61.67	59.53	3.48	−7.05	0.00	0.00	0.00	51.15
Genetic Algorithm	72.78	72.78	0.00	8.61	71.73	61.39	5.22	3.01	0.00	0.00	0.00	13.05
Simulated Annealing	63.06	63.06	0.00	−1.11	62.17	60.33	1.04	−6.55	0.00	0.00	0.00	17.17

Table 5. Performance measures of SVM optimized with meta-heuristic algorithms for God Class

Performance measures	Accuracy				ROC-AUC				*F*-measure			Time of one
Optimization Algorithms	Best [%]	Avg [%]	Std	Δ_1	Best [%]	Avg [%]	Std	Δ_2	Best [%]	Avg [%]	Std	iteration [s]
Arithmetic Optimization	71.67	71.67	0.00	−2.22	97.78	91.59	5.23	12.90	33.00	1.79	6.77	16.56
Jellyfish Search Optimization	93.89	90.32	9.66	20.00	95.11	94.95	0.54	10.23	92.29	92.29	0.00	33.53
Flow Direction Optimization	95.56	95.56	0.00	21.67	98.57	98.57	0.00	13.69	91.33	91.33	0.00	237.48
Student Psychology Based Optimization	94.03	91.14	7.66	20.14	**100.00**	**100.00**	0.00	**15.12**	89.24	71.39	35.70	29.86
Pathfinder Optimization	90.42	90.42	0.00	16.53	97.94	97.94	0.00	13.06	85.95	85.87	0.11	226.15
Sine Cosine Optimization	**98.75**	**98.75**	0.00	**24.86**	**100.00**	**100.00**	0.00	**15.12**	**98.57**	**98.57**	0.00	15.41
Jaya Optimization	93.75	84.90	11.80	19.86	**100.00**	**100.00**	0.00	**15.12**	91.24	58.39	43.79	13.98
Crow Search Optimization	78.75	70.94	1.76	4.86	95.56	81.78	5.45	10.68	57.00	6.95	15.14	20.07
Dragonfly Optimization	95.00	82.34	12.14	21.11	100.00	94.02	2.84	**15.12**	92.29	76.71	28.44	**9.17**
Krill Herd Optimization	59.72	59.72	0.00	−14.17	56.00	56.00	0.00	−28.88	0.00	0.00	0.00	24.75
Multi-Verse Optimization	95.14	80.67	12.52	21.25	95.44	93.96	1.21	10.56	93.24	50.13	44.50	27.49
Symbiotic Organisms Search Optimization	97.78	96.94	0.57	23.89	**100.00**	99.98	0.10	15.12	97.14	96.23	0.69	106.65
Flower Pollination Optimization	94.44	83.59	14.48	20.55	99.44	97.07	5.23	14.56	93.00	59.52	44.64	10.01
Teaching Learning Based Optimization	97.78	96.08	6.53	23.89	99.44	99.13	0.50	14.56	96.00	83.57	30.87	73.37
Gravitational Search Optimization	92.64	63.33	8.30	18.75	99.00	97.97	2.31	14.12	0.00	0.00	0.00	25.58
Biogeography-Based Optimization	93.06	67.63	7.50	19.17	97.56	95.50	3.40	12.68	90.86	21.65	38.53	30.62
Differential Evolution	67.92	67.92	0.00	−5.97	65.83	64.44	3.78	−19.05	0.00	0.00	0.00	16.41
Particle Swarm Optimization	69.17	69.17	0.00	−4.72	51.67	51.00	0.82	−33.21	0.00	0.00	0.00	35.02
Genetic Algorithm	65.42	65.42	0.00	−8.47	80.75	64.19	9.19	−4.13	0.00	0.00	0.00	14.94
Simulated Annealing	60.83	60.83	0.00	−13.06	66.33	60.05	3.80	−18.55	0.00	0.00	0.00	20.14

Optimization is the most proficient performer, consistently achieving the highest values across all three performance metrics. Sine Cosine Optimization attains a remarkable maximum accuracy of 98.75%, with the best average accuracy standing at 98.75%, as well. This impressive achievement represents a substantial improvement of 24.86% (Δ_1) compared to the non-optimized baseline. Furthermore, the algorithm achieves a maximum ROC-AUC of 100%, with the best average ROC-AUC reaching 100%. Notably, Student Psychology Based and Jaya Optimization also attain maximum and best average ROC-AUC values of 100%. Symbiotic Organisms Search Optimization also secures the best ROC-AUC value of 100%, effectively tying with its counterparts. Highest gain in ROC-AUC value observed is 15.12%. Regarding the F-measure, the maximum and best average value achieved is 98.57%. Notably, Krill Herd Optimization consistently ranks as the poorest-performing algorithm across all cases, exhibiting subpar results. Regarding computational efficiency, Dragonfly Optimization is the fastest algorithm in this context, with an execution time of 9.17 seconds per iteration. The results highlight Sine Cosine Optimization as the preeminent algorithm for detecting the God Class when combined with SVM.

4.1.4. Long method

Table 6 presents the findings of detecting Long Method using the Support Vector Machine (SVM) in conjunction with various meta-heuristic algorithms. Among these algorithms, **Symbiotic Organisms Search Optimization** stands out as the top-performing meta-heuristic, consistently exhibiting the highest values across all three performance metrics. Symbiotic Organisms Search Optimization attains an impressive maximum accuracy of 96.39%, a maximum ROC-AUC of 100%, and a maximum F-measure of 94.57%. It's also worth noting that Flower Pollination Optimization achieves a perfect ROC-AUC score of 100%. When considering the best average performance, Symbiotic Organisms Search Optimization secures the highest average accuracy of 95.34%, with a deviation of 0.75. Additionally, it achieves a best average ROC-AUC of 99.36% with a deviation of 0.40 and a best average F-measure of 93.50% with a deviation of 0.53. These findings underscore the algorithm's consistent and robust performance. Regarding improvements over the non-optimized baseline, Symbiotic Organisms Search Optimization achieves the maximum hike in accuracy and ROC-AUC, with increases of 32.22% (Δ_1) and 33.78% (Δ_2), respectively. Conversely, Krill Herd Optimization consistently ranks as the poorest-performing algorithm across all scenarios. Regarding computational efficiency, Dragonfly Optimization is the fastest technique, with an execution time of 8.17 seconds per iteration. To summarize, these results emphasize the superiority of Symbiotic Organisms Search Optimization for detecting Long Method when paired with SVM.

4.2. k-Nearest neighbors

Table 7 comprehensively presents the performance metrics encompassing the best and average values for accuracy, ROC-AUC, and F-measure concerning k-Nearest Neighbors (k-NN). The results are categorized into two scenarios: one when no optimization is applied and another when grid search is employed. In grid search, the hyperparameter spectrum is as follows – k: [ranges from 1 to 60], p: [1, 1.2, 1.5, 2]. In the absence of optimization, it is evident that the God Class stands out with the best accuracy of 71.81%, ROC-AUC of 68.25% and an F-measure of 49.33%. Upon the introduction of the grid search, the Long Method emerged as the leader in accuracy, achieving a notable 77.78%. Simultaneously, the

Table 6. Performance measures of SVM optimized with meta-heuristic algorithms for Long Method

Performance measures	Accuracy				ROC-AUC				F-measure			Time of one
Optimization Algorithms	Best [%]	Avg [%]	Std	Δ_1	Best [%]	Avg [%]	Std	Δ_2	Best [%]	Avg [%]	Std	iteration [s]
Arithmetic Optimization	89.17	72.67	6.09	25.00	98.61	86.32	7.11	32.39	64.33	2.77	12.60	18.61
Jellyfish Search Optimization	80.83	80.42	2.04	16.66	92.50	92.50	0.00	26.28	50.00	46.00	13.56	34.12
Flow Direction Optimization	91.94	90.92	0.30	27.77	97.22	97.22	0.00	31.00	88.38	86.70	0.62	228.81
Student Psychology Based Optimization	87.64	85.27	6.41	23.47	96.00	95.44	1.12	29.78	76.38	54.99	34.30	31.70
Pathfinder Optimization	87.22	87.22	0.00	23.05	97.00	97.00	0.00	30.78	84.88	84.88	0.00	229.40
Sine Cosine Optimization	87.08	87.08	0.00	22.91	97.50	97.50	0.00	31.28	65.00	65.00	0.00	16.19
Jaya Optimization	91.53	79.59	12.42	27.36	98.11	98.11	0.00	31.89	88.48	60.16	41.27	16.09
Crow Search Optimization	69.17	69.17	0.00	5.00	95.50	86.61	9.58	29.28	18.33	0.93	3.68	28.30
Dragonfly Optimization	94.03	82.51	10.45	29.86	97.33	92.87	2.64	31.11	91.14	76.57	22.64	**8.17**
Krill Herd Optimization	66.67	66.67	0.00	2.50	72.83	72.83	0.00	6.61	0.00	0.00	0.00	22.72
Multi-Verse Optimization	91.39	68.72	17.00	27.22	95.13	95.13	0.00	28.91	91.94	36.78	45.04	15.22
Symbiotic Organisms Search Optimization	**96.39**	**95.34**	0.75	**32.22**	**100.00**	**99.36**	0.40	**33.78**	**94.57**	**93.50**	0.53	82.00
Flower Pollination Optimization	90.42	77.42	12.49	26.25	**100.00**	96.90	9.68	**33.78**	87.43	66.45	37.34	9.72
Teaching Learning Based Optimization	94.03	94.03	0.00	29.86	99.33	98.89	0.16	33.11	92.13	77.39	33.77	75.38
Gravitational Search Optimization	71.67	71.67	0.00	7.50	97.22	95.11	4.45	31.00	36.67	1.47	7.19	19.51
Biogeography-Based Optimization	91.81	67.87	11.86	27.64	99.33	97.81	2.38	33.11	90.10	21.57	38.38	32.01
Differential Evolution	64.17	64.17	0.00	0.00	66.00	63.51	5.55	−0.22	0.00	0.00	0.00	10.36
Particle Swarm Optimization	67.92	67.92	0.00	3.75	64.33	62.29	3.10	−1.89	0.00	0.00	0.00	43.82
Genetic Algorithm	69.17	69.17	0.00	5.00	80.11	62.16	8.20	13.89	0.00	0.00	0.00	16.86
Simulated Annealing	70.42	70.42	0.00	6.25	63.50	60.47	0.91	−2.72	0.00	0.00	0.00	20.20

Table 7. Performance measures of k-Nearest Neighbors (k-NN)

	Without optimization			Grid search		
Code Smells	Accuracy	ROC-AUC	F-measure	Accuracy	ROC-AUC	F-measure
Data class	56.11	59.17	29.86	66.81	59.03	21.67
Feature envy	63.33	63.78	43.24	63.89	61.83	50.98
God class	71.81	68.25	49.33	74.17	67.33	51.86
Long method	68.06	65.62	15.67	**77.78**	65	42

God Class maintains prominence with the best ROC-AUC and F-measure values, amounting to 67.33% and 51.86%, respectively. While examining the magnitude of improvements by grid search, it becomes apparent that the uplift in performance measures is not particularly substantial. The most noteworthy enhancements include a 10.69% increase in accuracy for Data Class and 9.72% for Long Method. ROC-AUC always decreased if grid search is applied. A significant boost of 26.33% in F-measure for the Long Method is observed, but F-measure degraded by 8.19% for Data Class. In summary, the findings suggest that grid search, while functional, may not induce significant improvements in performance measures across all scenarios. Results indicate that there is a need for alternative strategy to boost performance.

4.2.1. Data class

The outcomes related to detecting the Data Class are thoughtfully presented in Table 8. Among the array of employed meta-heuristic algorithms, it's evident that **Flower Pollination Optimization** emerges as the most effective. It remarkably attains the maximum accuracy score of 100%, thereby exhibiting a substantial increase of 43.89% (Δ_1) compared to scenarios without optimization. Regarding average accuracy, Pathfinder seizes the top position, achieving a commendable accuracy of 97.62% with a negligible deviation of 0.05. Furthermore, for ROC-AUC values, Pathfinder, Sine Cosine, Jaya, Crow Search, Teaching Learning Based, Multi-Verse, and Flower Pollination Optimization jointly secure the highest value at 100%, reflecting an impressive hike of 40.83% (Δ_2). Pathfinder and Sine Cosine Optimization maintain this elevated performance level by achieving the best average ROC-AUC of 100%, with no deviations observed. Regarding the F-measure metric, Flower Pollination Optimization stands out, boasting a maximum value of 100% and a best average performance score of 96.45%. Finally, from an efficiency perspective, Dragonfly Optimization demonstrates its prowess by completing each iteration in a mere 4.90 seconds, rendering it the fastest among the considered optimization algorithms.

4.2.2. Feature envy

Table 9 presents the comprehensive results for detecting the Feature Envy with optimized k-NN. Remarkably, Symbiotic Organisms Search Optimization emerges as a standout performer, achieving the maximum accuracy and F-measure scores of 96.39% and 92.67%, respectively. Additionally, it boasts the best average accuracy and F-measure, securing impressive values of 96.30% and 92.32%. Notably, Symbiotic Organisms Search Optimization demonstrates a significant increase in accuracy, registering a top hike of 33.06% (Δ_1). Conversely, Multi-Verse Optimization excels in the ROC-AUC metric, showcasing the highest promenade of 35% (Δ_2). The ROC-AUC metric further reveals that Multi-Verse and Jaya Optimization jointly achieve the maximum and best average ROC-AUC scores at

Table 8. Performance measures of k-NN optimized with meta-heuristic algorithms for Data Class

Performance measures	Accuracy				ROC-AUC				F-measure			Time of one
Optimization Algorithms	Best [%]	Avg [%]	Std	Δ_1	Best [%]	Avg [%]	Std	Δ_2	Best [%]	Avg [%]	Std	iteration [s]
Arithmetic Optimization	95.28	92.47	2.16	39.17	99.33	98.87	0.27	40.16	94.29	90.56	2.79	40.59
Jellyfish Search Optimization	92.92	92.47	0.60	36.81	98.57	98.34	0.22	39.40	86.00	85.84	0.37	24.40
Flow Direction Optimization	96.25	96.20	0.24	40.14	99.58	99.37	0.21	40.41	95.14	95.12	0.11	266.47
Student Psychology Based Optimization	96.53	95.43	0.78	40.42	99.17	98.63	0.43	40.00	94.57	92.35	2.52	47.77
Pathfinder Optimization	97.64	**97.62**	0.05	41.53	**100.00**	**100.00**	0.00	**40.83**	95.71	95.28	0.35	186.47
Sine Cosine Optimization	95.42	94.88	0.64	39.31	**100.00**	**100.00**	0.00	**40.83**	94.29	93.77	0.54	14.72
Jaya Optimization	94.31	93.98	0.44	38.20	**100.00**	99.78	0.21	**40.83**	91.24	90.82	0.63	7.06
Crow Search Optimization	96.53	95.73	1.09	40.42	**100.00**	98.62	0.69	**40.83**	96.57	92.21	2.26	24.05
Dragonfly Optimization	96.25	93.67	1.57	40.14	99.58	98.72	1.02	40.41	92.67	89.16	2.70	**4.90**
Krill Herd Optimization	67.92	67.92	0.00	11.81	55.72	55.72	0.00	−3.45	0.00	0.00	0.00	38.23
Multi-Verse Optimization	94.17	93.18	1.23	38.06	**100.00**	99.78	0.18	**40.83**	93.21	92.35	0.90	13.71
Symbiotic Organisms Search Optimization	96.25	95.09	1.57	40.14	97.92	97.64	0.49	38.75	93.00	91.45	1.68	61.71
Flower Pollination Optimization	**100.00**	96.68	6.00	**43.89**	**100.00**	99.97	0.10	**40.83**	**100.00**	**96.45**	4.35	8.95
Teaching Learning Based Optimization	97.64	95.01	1.33	41.53	**100.00**	99.98	0.08	**40.83**	96.57	91.98	2.08	46.49
Gravitational Search Optimization	91.53	89.69	2.41	35.42	98.83	98.31	0.33	39.66	91.63	88.99	1.91	13.00
Biogeography-Based Optimization	95.14	94.37	0.83	39.03	99.44	97.84	0.67	40.27	91.67	89.74	2.06	27.12
Differential Evolution	77.50	70.93	5.07	21.39	84.50	77.03	5.22	25.33	66.19	61.22	5.42	8.33
Particle Swarm Optimization	83.19	80.14	3.69	27.08	87.36	86.55	0.94	28.19	81.35	77.25	4.44	7.31
Genetic Algorithm	79.86	75.53	2.81	23.75	86.39	81.49	2.67	27.22	69.19	57.34	4.50	26.45
Simulated Annealing	78.47	75.73	2.22	22.36	81.94	80.59	0.83	22.77	69.81	59.71	15.37	17.41

Table 9. Performance measures of k-NN optimized with meta-heuristic algorithms for Feature Envy

Performance measures	Accuracy				ROC-AUC				F-measure			Time of one
Optimization algorithms	Best [%]	Avg [%]	Std	Δ_1	Best [%]	Avg [%]	Std	Δ_2	Best [%]	Avg [%]	Std	iteration [s]
Arithmetic Optimization	91.81	86.67	3.51	28.48	96.17	93.72	1.14	32.39	80.67	74.26	9.51	18.81
Jellyfish Search Optimization	88.19	88.05	0.51	24.86	98.33	98.07	0.23	34.55	75.67	73.96	73.96	32.04
Flow Direction Optimization	87.22	86.80	0.99	23.89	95.56	95.39	0.20	31.78	75.24	74.07	1.55	285.53
Student Psychology Based Optimization	88.33	83.99	3.63	25.00	95.83	93.62	1.26	32.05	70.67	56.63	8.68	49.74
Pathfinder Optimization	94.17	94.13	0.19	30.84	98.47	98.03	0.22	34.69	88.17	88.17	0.00	193.51
Sine Cosine Optimization	90.69	90.67	0.05	27.36	96.00	95.49	0.27	32.22	79.57	79.40	0.44	10.53
Jaya Optimization	90.69	90.50	0.45	27.36	98.75	**98.11**	0.30	34.97	79.00	78.59	0.86	5.91
Crow Search Optimization	83.33	80.06	3.45	20.00	94.61	93.24	0.74	30.83	63.00	47.13	14.07	25.11
Dragonfly Optimization	85.83	82.16	2.89	22.50	96.67	95.19	0.98	32.89	67.00	54.67	11.83	**4.27**
Krill Herd Optimization	74.86	70.57	2.03	11.53	78.67	74.72	1.68	14.89	48.72	39.52	6.13	9.61
Multi-Verse Optimization	89.31	88.69	1.01	25.98	**98.78**	97.42	0.55	**35.00**	80.00	78.50	2.52	11.45
Symbiotic Organisms Search Optimization	**96.39**	**96.30**	0.30	**33.06**	98.19	97.52	1.50	34.41	**92.67**	**92.32**	0.61	76.47
Flower Pollination Optimization	94.03	91.57	1.65	30.70	98.50	97.33	1.04	34.72	89.81	74.44	21.30	9.05
Teaching Learning Based Optimization	92.92	91.00	1.90	29.59	97.58	96.77	0.77	33.80	90.38	87.76	2.34	47.05
Gravitational Search Optimization	88.19	86.73	2.56	24.86	96.58	94.98	0.91	32.80	82.48	79.49	7.66	17.54
Biogeography-Based Optimization	90.00	86.07	2.88	26.67	95.67	94.78	0.46	31.89	81.63	77.94	4.76	33.56
Differential Evolution	77.50	74.86	2.74	14.17	83.39	81.83	1.33	19.61	61.50	50.18	9.69	10.79
Particle Swarm Optimization	74.03	73.33	0.62	10.70	81.11	80.78	0.35	17.33	51.43	49.78	0.91	7.21
Genetic Algorithm	75.42	73.42	1.73	12.09	82.97	79.57	2.96	19.19	45.00	28.24	14.53	33.69
Simulated Annealing	74.17	71.87	1.41	10.84	76.81	75.51	0.96	13.03	35.67	21.11	11.39	12.22

98.78% and 98.11%, respectively, with a minimal deviation of 0.30. From an operational efficiency standpoint, Dragonfly Optimization exhibits remarkable swiftness, completing each iteration in a mere 4.27 seconds, thereby asserting itself as the fastest-performing algorithm in the context of this study. In conclusion, optimizing k-NN with **Symbiotic Organisms Search Optimization** is the most effective approach for detecting the Feature Envy.

4.2.3. God class

The results for detecting the God Class are meticulously outlined in Table 10. Notably, Flow Direction Optimization delivers outstanding performance, clinching the maximum accuracy, best average accuracy, and the highest hike in accuracy, reaching impressive scores of 97.64%, 97.64% (with zero deviations), and a remarkable 25.83% hike (Δ_1). Moreover, the ROC-AUC metric showcases exceptional results, with a maximum ROC-AUC score of 100% and the most significant hike, reaching 31.75% (Δ_2). These outstanding achievements are credited to Arithmetic, Jellyfish Search, Flow Direction, Pathfinder, Jaya, Crow Search, Multi-Verse, Symbiotic Organisms Search, Flower Pollination, Teaching Learning Based, Gravitational Search, and Biogeography-Based Optimization. In addition, Jellyfish Search, Flow Direction, Pathfinder, Multi-Verse, Teaching Learning Based, and Biogeography-Based Optimization collectively secure a best average ROC-AUC of 100%, accompanied by zero deviations. Regarding the F-measure, Flow Direction Optimization stands out with a maximum and average value of 96%, with no deviations. In terms of operational efficiency, Jaya Optimization demonstrates impressive speed, completing each iteration in 5.08 seconds. Additionally, it's worth noting that both Differential Evolution and Simulated Annealing achieve perfect scores of 100% for both best and average ROC-AUC values. In conclusion, utilizing k-NN in conjunction with **Flow Direction Optimization** is the most effective approach to detecting the God Class despite a slightly slower execution time.

4.2.4. Long method

Table 11 comprehensively presents the outcomes concerning detecting the Long Method code. Notably, Biogeography-Based Optimization stands out with the maximum accuracy, the best average accuracy, and the most significant accuracy hike, attaining remarkable scores of 96.39%, 96.39%, and 28.33% hike (Δ_1), respectively. Moreover, the ROC-AUC metric showcases exceptional results, with a maximum ROC-AUC score of 100% and the most substantial hike, reaching 34.38% (Δ_2). These exceptional achievements are attributed to Jellyfish Search, Student Psychology Based, Sine Cosine, Jaya, Crow Search, Multi-Verse, Symbiotic Organisms Search, Flower Pollination, Teaching Learning Based, Gravitational Search, and Biogeography-Based Optimization. Furthermore, Jellyfish Search, Student Psychology Based, and Symbiotic Organisms Search Optimization collectively secure the best average ROC-AUC of 100%. Regarding the F-measure, Biogeography-Based Optimization achieves the maximum value of 94%, while Multi-Verse Optimization secures the best average F-measure of 88.44%. Dragonfly Optimization is the fastest, completing each iteration in 5.84 seconds. It's worth highlighting that the other elementary algorithms also deliver commendable performance in this context. In summary, when it comes to detecting the Long Method, using k-NN in conjunction with **Biogeography-Based** or

Table 10. Performance measures of k-NN optimized with meta-heuristic algorithms for God Class

Performance measures	Accuracy				ROC-AUC				F-measure			Time of one
Optimization Algorithms	Best [%]	Avg [%]	Std	Δ_1	Best [%]	Avg [%]	Std	Δ_2	Best [%]	Avg [%]	Std	iteration [s]
Arithmetic Optimization	93.89	88.99	5.47	22.08	**100.00**	99.84	0.27	**31.75**	86.00	70.50	15.13	33.46
Jellyfish Search Optimization	93.89	93.84	0.22	22.08	**100.00**	**100.00**	0.00	**31.75**	87.67	87.19	1.30	29.67
Flow Direction Optimization	**97.64**	**97.64**	0.00	**25.83**	**100.00**	**100.00**	0.00	**31.75**	**96.00**	**96.00**	0.00	240.65
Student Psychology Based Optimization	94.03	92.92	1.24	22.22	98.61	98.51	0.16	30.36	87.33	85.03	5.18	47.80
Pathfinder Optimization	92.92	92.92	0.00	21.11	**100.00**	**100.00**	0.00	**31.75**	91.43	90.81	0.60	185.17
Sine Cosine Optimization	90.69	90.69	0.00	18.88	99.67	99.27	0.19	31.42	82.67	82.67	0.00	12.47
Jaya Optimization	96.39	94.99	0.79	24.58	**100.00**	99.73	0.29	**31.75**	90.00	89.27	1.19	**5.08**
Crow Search Optimization	91.53	87.58	3.61	19.72	**100.00**	99.76	0.31	**31.75**	81.00	73.71	10.30	17.61
Dragonfly Optimization	92.92	85.54	6.06	21.11	99.44	98.68	0.45	31.19	82.67	60.81	22.08	5.57
Krill Herd Optimization	72.36	67.23	2.60	0.55	85.23	78.59	3.37	16.98	51.79	40.31	7.24	10.90
Multi-Verse Optimization	94.03	93.53	0.56	22.22	**100.00**	**100.00**	0.00	**31.75**	91.24	90.25	1.13	13.27
Symbiotic Organisms Search Optimization	91.81	91.57	0.43	20.00	**100.00**	99.99	0.05	**31.75**	82.33	79.56	2.51	62.28
Flower Pollination Optimization	93.89	88.12	5.63	22.08	**100.00**	99.64	0.33	**31.75**	89.00	81.72	7.94	10.41
Teaching Learn ing Based Optimization	88.06	86.82	1.61	16.25	**100.00**	**100.00**	0.00	**31.75**	80.71	74.73	6.24	46.68
Gravitational Search Optimization	95.42	93.20	1.62	23.61	**100.00**	99.93	0.16	**31.75**	93.71	90.12	4.46	17.06
Biogeography-Based Optimization	92.64	87.37	6.18	20.83	**100.00**	**100.00**	0.00	**31.75**	79.33	61.85	19.78	66.86
Differential Evolution	96.39	95.79	0.94	24.58	100.00	100.00	0.00	31.75	92.67	90.75	4.02	10.06
Particle Swarm Optimization	96.53	96.53	0.00	24.72	100.00	99.96	0.11	31.75	92.67	92.67	0.00	6.88
Genetic Algorithm	96.39	94.17	0.72	24.58	100.00	99.77	0.26	31.75	93.14	90.45	2.29	33.48
Simulated Annealing	96.39	93.43	3.92	24.58	100.00	99.33	1.33	31.75	93.00	87.15	10.04	15.01

Table 11. Performance measures of k-NN optimized with meta-heuristic algorithms for Long Method

Performance measures	Accuracy				ROC-AUC				F-measure			Time of one
Optimization Algorithms	Best [%]	Avg [%]	Std	Δ_1	Best [%]	Avg [%]	Std	Δ_2	Best [%]	Avg [%]	Std	iteration [s]
Arithmetic Optimization	88.33	80.65	4.68	20.27	99.67	99.25	0.25	34.05	81.81	69.82	10.86	15.23
Jellyfish Search Optimization	90.69	90.36	0.85	22.63	**100.00**	**100.00**	0.00	**34.38**	84.24	83.62	1.29	30.59
Flow Direction Optimization	84.44	84.44	0.00	16.38	96.94	96.72	0.27	31.32	63.00	63.00	0.00	272.84
Student Psychology Based Optimization	90.69	89.53	2.63	22.63	**100.00**	**100.00**	0.00	**34.38**	84.90	80.72	5.87	31.00
Pathfinder Optimization	84.31	83.89	0.66	16.25	99.67	99.39	0.36	34.05	68.67	68.67	0.00	203.61
Sine Cosine Optimization	89.44	89.14	0.53	21.38	**100.00**	99.92	0.13	**34.38**	79.00	78.43	1.02	28.44
Jaya Optimization	84.86	82.86	1.05	16.80	**100.00**	99.26	0.84	**34.38**	76.71	74.30	1.86	14.56
Crow Search Optimization	87.78	81.33	4.72	19.72	**100.00**	99.29	0.47	**34.38**	70.33	58.02	11.43	27.80
Dragonfly Optimization	88.06	83.47	2.12	20.00	99.29	98.12	0.68	33.67	50.00	18.40	13.60	**5.84**
Krill Herd Optimization	89.17	80.89	5.65	21.11	97.50	92.91	2.43	31.88	80.57	66.27	10.47	7.82
Multi-Verse Optimization	94.17	**93.88**	0.42	26.11	**100.00**	99.73	0.24	**34.38**	89.00	**88.44**	0.59	31.89
Symbiotic Organisms Search Optimization	90.42	89.38	0.93	22.36	**100.00**	**100.00**	0.00	**34.38**	71.67	68.65	2.76	69.10
Flower Pollination Optimization	88.89	83.81	6.06	20.83	**100.00**	99.38	0.30	**34.38**	76.57	67.91	10.68	28.75
Teaching Learning Based Optimization	91.67	90.16	1.74	23.61	**100.00**	99.98	0.08	**34.38**	86.24	83.22	4.89	66.39
Gravitational Search Optimization	89.31	87.77	1.31	21.25	**100.00**	99.82	0.25	**34.38**	82.24	75.16	7.31	35.45
Biogeography-Based Optimization	**96.39**	90.43	6.41	**28.33**	**100.00**	99.68	0.20	**34.38**	**94.00**	83.42	12.64	65.35
Differential Evolution	86.67	84.17	1.57	18.61	96.33	94.94	0.91	30.71	69.00	61.91	13.58	11.08
Particle Swarm Optimization	87.08	86.06	0.30	19.02	95.00	93.77	2.21	29.38	69.67	63.29	3.58	22.47
Genetic Algorithm	86.94	80.01	6.70	18.88	94.03	91.60	1.33	28.41	76.00	61.67	19.06	47.63
Simulated Annealing	94.03	93.21	0.98	25.97	98.11	98.04	0.24	32.49	92.07	91.03	1.51	31.66

Multi-Verse Optimization emerges as the most effective approach, offering excellent accuracy and ROC-AUC outcomes.

5. Discussion

The following section addresses the research questions and discusses the important findings of the research study.

5.1. Does using meta-heuristic algorithms for optimizing machine learning classifiers boost their performance to detect code smell in complex software systems?

The described experiment has been executed to address RQ 1, yielding noteworthy advancements in performance metrics. In the case of Support Vector Machine (SVM), the highest accuracy is 98.75%, achieved by the Sine Cosine Optimization algorithm when applied to detect God Class. The Symbiotic Organisms Search elevates the accuracy by 32.22% when deployed for Long Method. Regarding ROC-AUC metrics, a flawless 100% is reached by Symbiotic Organisms Search across multiple code smell detection, specifically for Data Class, God Class, and Long Method.

In the case of God Class, the ROC-AUC value of 100% is also secured by Student Psychology Based, Sine Cosine, and Jaya Optimization methods. Similarly, Flower Pollination Optimization achieves an impeccable 100% ROC-AUC for Long Method. The average ROC-AUC stands at 100% and is simultaneously attained by Student Psychology Based, Sine Cosine, and Jaya Optimization for God Class. For detecting Data Class, Symbiotic Organisms Search orchestrates a 45.11% hike in ROC-AUC value.

Turning our focus to F-measure, the Sine Cosine Optimization achieved 98.57% to combat God Class. Dragonfly Optimization is acknowledged as the fastest in algorithmic velocity, while Pathfinder Optimization is the slowest. Conclusively, the apex of optimization is occupied by the **Sine Cosine Algorithm**, demonstrably exemplifying its pre-eminence by securing the highest scores, both in terms of maximum and average values, across a spectrum of performance metrics.

For k-Nearest Neighbors (k-NN), a perfect 100% accuracy and 43.89% surge in accuracy is recorded, executed by the Flower Pollination Optimization method when applied to detect Data Class. It also scores a perfect 100% F-measure and the best average F-measure at 96.45%. When applied to detect God Class, the highest average accuracy is 97.64% attained by the Flow Direction Optimization algorithm.

For ROC-AUC metrics, a flawless 100% is not a solitary accomplishment but a shared distinction among several optimization methodologies. Specifically, optimizers for Data Class include Pathfinder, Sine Cosine, Jaya, Crow Search, Multi-Verse, Flower Pollination, and Teaching Learning Based Optimization, concurrently ascending to this pinnacle. Each eminent algorithm also accomplishes a 40.83% ROC-AUC increase. Similarly, the God Class bears witness to the Arithmetic, Jellyfish Search, Flow Direction, Pathfinder, Jaya, Crow Search, Multi-Verse, Symbiotic Organisms Search, Flower Pollination, Teaching Learning Based, Gravitational Search, and Biogeography-Based Optimization, all attaining a flawless 100% ROC-AUC value. The Long Method equally experiences perfection in ROC-AUC, with Jellyfish Search, Student Psychology Based, Sine Cosine, Jaya, Crow Search, Multi-Verse, Symbiotic Organisms Search, Flower Pollination, Teaching Learning

Based, Gravitational Search, and Biogeography-Based Optimization, all registering a 100% ROC-AUC.

The average ROC-AUC yields a harmonious 100% outcome with no deviations for several scenarios: Pathfinder and Sine Cosine Optimization for Data Class, Jellyfish Search, Flow Direction, Pathfinder, Multi-Verse, Teaching Learning Based, and Biogeography-Based Optimization for God Class, and Jellyfish Search, Student Psychology Based, and Symbiotic Organisms Search Optimization for Long Method. Regarding the computational time, Dragonfly Optimization is the fastest, while the Flow Direction Optimization method is the slowest in its computational stride. The paramount optimizer, defined by maximum and average performance measures, emerges as the **Flower Pollination Optimization**, underscoring its dominance in k-NN optimization.

Summary of RQ 1. Employing swarm-based techniques to optimize the hyperparameter values of machine learning classifiers is definitely a beneficial process. It not only improves the performance of a classifier but eliminates the need for an expert, automating the code smell detection process.

5.2. How significant is the impact of optimization of machine learning algorithms with meta-heuristic techniques on its overall performance?

To answer RQ 2, we have conducted statistical tests on experiment results to evaluate the impact of optimization. The Wilcoxon signed-rank test is a non-parametric statistical test used to assess whether the distribution of paired differences between two related groups is symmetric about zero [96]. Experimentation data do not follow a normal distribution, have paired observations, and data can be ranked. Therefore, the Wilcoxon signed-rank test is the best hypothesis statistical test to measure the impact of employing meta-heuristic algorithms for optimizing machine learning algorithms.

To perform the test, a null hypothesis (H0) is set up as – the median difference between paired observations is zero (no difference) and the alternative hypothesis (H1) as the median difference between paired observations is not zero. Data is gathered for the paired observations we want to compare, and the differences between paired observations are calculated. The absolute values of the differences are ranked, and the test statistic (W) using the ranked differences is calculated. For n pairs, the degree of freedom is $n-1$. The test statistic (p-value) to the critical value from the Wilcoxon signed-rank distribution table is compared. If the p-value is less than the chosen significance level, reject the null hypothesis, indicating a significant difference. If the p-value is greater than the significance level, fail to reject the null hypothesis. The test statistic, degrees of freedom, p-value, and decision regarding the null hypothesis are reported [97].

Tables 12–15 results depict the value of z, p, and r from the Wilcoxon signed rank sum test. Values before optimization are paired with best and average values acquired after optimization. The degree of freedom for this test is 15. The confidence level is 95%, and the significance level is 0.05. The null hypothesis is rejected if the p-value is less than 0.05, implying the difference is significant. Based on the results, it can be seen that the p-values for all five performance measures are below 0.05, indicating a significant difference between performance measures before and after optimization. r denotes effect size depicting the magnitude of difference and can be calculated as $\frac{z}{\sqrt{n}}$, where n is the number of paired

Table 12. Wilcoxon Signed Rank Sum test results for Best Values attained by SVM

Performance measures	Accuracy			*F*-measure			ROC-AUC		
Code smell	z	p	r	z	p	r	z	p	r
Data class	3.5180	.000435	0.88	3.5168	.000437	0.88	3.5174	.000436	0.88
Feature envy	3.5168	.000437	0.88	3.4651	.000530	0.87	3.3616	.000775	0.84
God class	3.3099	.000933	0.83	2.6983	.006969	0.67	3.2966	.000979	0.82
Long method	3.5168	.000437	0.88	3.5180	.000435	0.88	3.4128	.000643	0.85

Table 13. Wilcoxon Signed Rank Sum test tesults for Average Values attained by SVM

Performance measures	Accuracy			*F*-measure			ROC-AUC		
Code smell	z	p	r	z	p	r	z	p	r
Data class	3.5162	.000438	0.88	3.5162	.000438	0.88	3.5168	.000437	0.88
Feature envy	3.3611	.000776	0.84	3.4133	.000642	0.85	2.7923	.005234	0.70
God class	2.3786	.017378	0.59	2.6389	.008317	0.66	3.2958	.000982	0.82
Long method	3.5162	.000438	0.88	3.5162	.000438	0.88	2.7923	.005234	0.70

Table 14. Wilcoxon Signed Rank Sum test results for Best Values attained by k-NN

Performance measures	Accuracy			*F*-measure			ROC-AUC		
Code smell	z	p	r	z	p	r	z	p	r
Data class	3.5197	.000432	0.88	3.4980	.000469	0.87	3.4656	.000529	0.87
Feature envy	3.5174	.000436	0.88	3.5162	.000438	0.88	3.5162	.000438	0.88
God class	3.5197	.000432	0.88	3.6973	.000218	0.92	3.5168	.000437	0.88
Long method	3.5168	.000437	0.88	3.6537	.000258	0.91	3.5162	.000438	0.88

Table 15. Wilcoxon Signed Rank Sum test results for Average Values attained by k-NN

Performance measures	Accuracy			*F*-measure			ROC-AUC		
Code smell	z	p	r	z	p	r	z	p	r
Data class	3.5168	.000437	0.88	3.4656	.000529	0.87	3.4651	.000530	0.87
Feature envy	3.5162	.000438	0.88	3.5162	.000438	0.88	3.4645	.000531	0.87
God class	3.4651	.000530	0.87	3.5369	.000405	0.88	3.4645	.000531	0.87
Long method	3.5162	.000438	0.88	3.5185	.000434	0.88	3.5162	.000438	0.88

observations. The effect is considered high if r is greater than 0.5 and 0.8 is recorded r value in the experimentation, yielding promising results.

Summary of RQ 2. Optimizing machine learning algorithms with swarm-intelligent algorithms significantly impacts their performance.

5.3. Given the meta-heuristic algorithms, which yields the best performance in optimizing classifiers to detect code smell and why?

To address RQ 3, the experiment's outcomes are examined and compared to determine the most effective meta-heuristic techniques for optimizing machine learning algorithms for code smell detection. It is important to acknowledge that the "No-Free Lunch" theorem has significantly influenced the landscape of optimization algorithms, driving continuous

innovations over the years. This theorem underscores that no single algorithm universally excels in every problem domain. Instead, their efficacy varies, with each demonstrating superior performance in specific problem statements [98]. Implementation remains the most effective means of identifying the optimal technique for a given problem.

The investigation involved systematically applying sixteen meta-heuristic algorithms for hyperparameter optimization on two distinct machine learning algorithms, enhancing their performance metrics. A comprehensive evaluation is conducted across three scenarios: instances without optimization, cases employing grid search, and evaluations utilizing meta-heuristic algorithms. Performance metrics are thoughtfully juxtaposed with other algorithms, ensuring an accurate and effective comparison. These assessments are supplemented by comparisons with foundational algorithms, namely Genetic Algorithm, Differential Evolution, Particle Swarm Optimization, and Simulated Annealing. The empirical results affirm that the foundational algorithms, while competent, do not outshine the implemented optimizers across the board. Instead, they exhibit comparable proficiency in a few cases. The comprehensive evaluation of their performance measures, in conjunction with other algorithms, is methodically documented after the respective tables. Table 16 highlights the highest-performing optimization algorithms for each case based on experimentation.

Table 16. The best performing optimization algorithm for each code smell

Code Smell	SVM	k-NN
Data class	Symbiotic Organisms Search Optimization	Flower Pollination Optimization
Feature envy	Dragonfly Optimization	Symbiotic Organisms Search Optimization
God class	Sine Cosine Optimization	Flow Direction Optimization
Long method	Symbiotic Organisms Search Optimization	Biogeography-Based and Multi-Verse Optimization

Finding the most optimized value for hyperparameters of machine learning algorithms is in the category of non-separable, constrained, and multimodal problems. Non-separable problems refer to scenarios where the relationships and dependencies within the data are too intricate to be accurately represented by simple linear decision boundaries [99]. In classification tasks, linear separability implies that classes can be perfectly distinguished by a straight line, plane, or hyperplane, but non-separable problems defy such simplicity. Dealing with non-separable data requires complex decision boundaries, often necessitating the application of nonlinear models like kernelized support vector machines. Specialized algorithms like meta-heuristics or evolutionary approaches may be needed to navigate such landscapes.

Constrained problems refer to scenarios where the solution space of a problem is subject to certain conditions or limitations [100]. These constraints restrict the set of feasible solutions and play a critical role in shaping the optimization landscape. Optimization algorithms designed for constrained problems must navigate the complex interplay between the objective function and the imposed constraints. Classical optimization methods, like Lagrange multipliers and penalty methods, are often employed to handle equality and inequality constraints.

Multimodal problems refer to scenarios where the objective function or fitness landscape has multiple distinct optimal solutions, known as modes [101]. Each mode represents a set of parameter values that yield an optimal or near-optimal solution to the problem where the algorithm can converge. The presence of multiple modes introduces challenges because traditional optimization methods, which aim to find a single global optimum, may struggle

to explore and exploit the diverse modes. Handling multimodal problems requires specialized optimization techniques designed to explore and exploit multiple modes efficiently.

Meta-heuristic algorithms are easy to implement and do not require much domain-specific knowledge. They can optimize both continuous, discrete problems, and multiple objective problems. The performance of a meta-heuristic algorithm hinges upon some of the pivotal factors. The first factor is the precise configuration of parameters, encompassing critical attributes such as the optimal count of search agents, the judicious establishment of the discovery rate, the velocity of the fitness function, etc. Furthermore, a paramount significance revolves around the delicate equilibrium between exploration and exploitation rates. Though they can find the global optima for complex, nonlinear, and non-convex functions, they are prone to get trapped in local optima if the population size is small and the search space is vast. Augmenting this complexity, randomness into the search process emerges as a potent mechanism. By introducing controlled stochasticity, these optimizers foster heightened performance and enhanced exploration of solution spaces, ultimately yielding superior results [102]. The **Symbiotic Organisms Search Optimization, Teaching Learning Based Optimization, and Sine Cosine Optimization** emerge as stellar exemplars of detecting code smells, adeptly navigating the intricate terrain of algorithmic design. They orchestrate a harmonious symphony of parameter tuning, dynamic mode switching, and controlled randomness infusion, culminating in attaining superlative outcomes. Conversely, Krill Herd Optimization is the least effective algorithm for code smell detection.

Summary of RQ 3. The no-free Lunch theorem implies that there is no one-size-fits-all solution; what works best for one optimization problem might not work for another problem. So, the best way to find the most optimal techniques is to implement them and compare their results. Table 16 summarizes the list of best-performing optimization techniques for each case. They performed better because they balanced exploration and exploitation well, avoided early convergence, introduced appropriate randomness, and discovered global optimum solutions required to conquer non-separable, constrained, and multi-modal problems.

5.4. How does our approach perform compared to existing machine learning based techniques?

To answer RQ 4, we have compared our work with Fontana et al. [46]. This is the most extensive study that detects code smells using machine learning and employs the same datasets, allowing for a fair comparison. They created balanced datasets to detect the four most common and perilous code smells. They applied 32 variations of machine learning classifiers, including their boosted versions, for detection. It included pruned, unpruned, and reduced error pruning techniques of J48, a C4.5 decision tree. JRip, Random Forest, Naive Bayes, and SMO with RBF and Polynomial kernel were also included. With that, C and ν SVM were implemented with Linear, Polynomial, RBF, and Sigmoid kernel settings. Implementation was done in Weka, and machine learning classifiers were treated as black-box implementations. No pre-processing or feature selection technique was used except in the case of SVM, where standardization and normalization were done. They employed 10-fold cross-validation techniques and reported average values. Tree-based algorithms like J48 and random forest performed best, whereas SVMs were the worst performers.

For Data Class, C-SVM with RBF has an accuracy of 96%, F-measure of 97.01%, and ROC-AUC of 99.15%. Symbiotic Organisms Search Optimization is the best performer

in detecting Data Class with SVM and outperformed accuracy and AUC delivered by Fontana et al., reported accuracy is 97.64%, F-measure is 96%, and ROC-AUC is 100%. In the case of Feature Envy, results projected by Fontana et al. were far less superior. The dragonfly optimizer performed best with 96.25% accuracy, 94.29% F-measure, and 99.33% ROC-AUC, whereas Fontana et al. yielded 94.14% accuracy, 95.62% F-measure, and 98.02% ROC-AUC. For God Class, the Sine Cosine algorithm outperformed all the above-mentioned meta-heuristic algorithms and results of Fontana et al. Accuracy, F-measure, and ROC-AUC achieved for God Class in the case of Fontana et al. are 95.76%, 96.87%, and 99.24%, respectively, whereas optimized SVM achieved 98.75% accuracy, 98.57% F-measure, and 100% ROC-AUC, all on the higher side. Symbiotic Organisms Search Optimization obtained 96.39% accuracy and 100% ROC-AUC for the Long Method, which is higher compared to the performance measured attained by Fontana et al., i.e., 96.38% accuracy and 99.15% ROC-AUC. One exception is the F-measure, which is 94.57% for optimized SVM and 97.22% for the unoptimized version.

Summary of RQ4. Unlike F-measure in two out of four cases, utilizing swarm-based algorithms for optimizing SVM is a better option as it delivers elevated performance.

6. Threats to validity

In this section, threats to validity are discussed that might arise concerns and how they are mitigated.

6.1. Threats to internal validity

The assessment of metrics within the datasets [46] is conducted using a proprietary tool known as Design Features and Metrics for Java (DFMC4J). This tool operates by parsing Java code through the Eclipse JDT Library; however, it is important to note that the accuracy of its calculations has not been externally validated, potentially introducing imprecision in metric computations for source code elements. Moreover, the identification of code smell candidates is carried out manually by students rather than seasoned professionals, thereby introducing an inherent margin of error. To mitigate this concern, a comprehensive training program was administered to the students, and the final decisions were made following meticulous deliberation. In addition to this, code smell detection tools like iPlasma, PMD, and Fluid tools were also enlisted to corroborate the presence of code smell instances.

6.2. Threats to external validity

The datasets were meticulously crafted from a collection of 74 open-source Java systems sourced from the Qualitus Corpus [67]. Nonetheless, it's imperative to acknowledge that open-source software might not encompass the entirety of conceivable scenarios, potentially limiting the generalization of findings to industrial contexts. Extending our investigation to encompass industrial, commercial, and private projects is a future endeavor. These systems employ older Java versions and don't include emerging new Java language constructs [103]. The systems included are from 2003-2011, which might not represent the current scenario. These issues can be addressed in future work by employing datasets that include the latest Java constructs, industrial projects, more code smells, severity prospects, etc.

It is important to underscore that this empirical study focuses solely on datasets originating from Java source code, and its findings may not seamlessly translate to other programming languages given the distinct nature of metric values and design paradigms across languages. As part of our ongoing research, we aim to explore additional programming languages to augment the breadth of our insights. Furthermore, while Fowler [6] delineates twenty-two distinct code smells, this study delves into the analysis of only four, a limited subset for generalization. Future investigations could encompass the remaining smells to yield more comprehensive and conclusive outcomes. Similarly, the utilization of merely two classifiers in this study warrants consideration, as the implications derived may not be universally applicable.

6.3. Threats to conclusion validity

The computation of F-measure by certain machine learning algorithms faced limitations, impacting the derived conclusions concerning F-measure values. This issue is of considerable significance, warranting both immediate attention and subsequent in-depth investigation. While maintaining nearly identical parameters, including population size and generations, across various meta-heuristic algorithms facilitates fair comparisons, it's worth noting that these parameters might inadvertently affect certain algorithms due to their diverse search agent requirements. Additionally, the uniformity of stopping criteria is 50 iterations for each algorithm might not ensure fairness, given the inherent variability in convergence rates among different algorithms.

7. Conclusions and future work

Our investigation delves into the merits of diverse meta-heuristic algorithms as tools for optimizing supervised machine learning techniques. Additionally, we have conducted a comparative analysis of results between machine learning classifiers, both pre and post optimization. The findings from our study are summarized as follows:

1. The top-performing meta-heuristic algorithm is Symbiotic Organisms Search Optimization. Conversely, Krill Herd Optimization exhibited the lowest performance in the context of code smell detection.
2. In the case of Support Vector Machine, the apex metrics include an accuracy rate of 98.75%, a perfect ROC-AUC score of 100%, and an F-measure of 98.57%. The maximum improvement in accuracy and ROC-AUC observed is 32.22% and 45.11%, respectively.
3. The best k-Nearest Neighbor, outcomes are marked by a flawless accuracy rate, ROC-AUC, and F-measure value of 100%. The accuracy and ROC-AUC surged by 43.89% and 40.83%, respectively, through applying optimization algorithms.
4. SVM showcased its optimum performance when coupled with Sine Cosine Optimization, whereas k-NN exhibited superior results when joined with Flower Pollination Optimization.
5. A Rigorous statistical test underscores the profound impact of meta-heuristic algorithms in fine-tuning hyperparameters of machine learning algorithms, thereby enhancing their overall performance.
6. SVM excels in detecting God Class and k-NN masters in identifying Data Class instances, all achieved through optimizing machine learning classifiers via meta-heuristic algorithms.

Here are the prospective avenues for further research stemming from this study:

1. Future work could utilize improved versions of meta-heuristic algorithms, characterized by improved convergence speed, exploration capabilities, and diminished sensitivity to hyperparameters.
2. Exploring multi-objective, binary, hybrid, chaotic and alternative variants of meta-heuristic techniques hold the premise for achieving heightened efficiency, adaptability and flexibility in optimization processes.
3. An extensive array of over two hundred meta-heuristic algorithms exist, whereas our study has selectively implemented specific types. Future research endeavors could extend to comparative assessments with a broader spectrum of optimization algorithms.
4. The implementation and evaluation of novel optimization algorithms, including but not limited to Central Force Optimization, Vortex Search Algorithm, Thermal Exchange Optimization, and Artificial Electric Field Algorithm, offer intriguing prospects for further inquiry.
5. Expanding the scope to optimize various other machine learning classifiers such as Random Forest, Decision Tree, JRip, and Naive Bayes, among others, holds potential for diversifying the application domains of these techniques.
6. The drive to optimize machine learning classifiers for detecting various other code smells or anti-patterns presents an engaging research avenue.
7. Investigating code smells in programming languages beyond Java constitutes a compelling direction for future research, broadening the applicability of the findings.
8. The exploration of feature engineering and selection methodologies utilizing meta-heuristic algorithms emerges as an avenue with the potential to augment the performance of machine learning classifiers.

References

[1] I. Ozkaya, "The next frontier in software development: AI-augmented software development processes," *IEEE Software*, Vol. 40, No. 4, 2023, pp. 4–9.

[2] H.J. Christanto and Y.A. Singgalen, "Analysis and design of student guidance information system through software development life cycle (SDLC) and waterfall model," *Journal of Information Systems and Informatics*, Vol. 5, No. 1, 2023, pp. 259–270.

[3] M. Almashhadani, A. Mishra, A. Yazici, and M. Younas, "Challenges in agile software maintenance for local and global development: An empirical assessment," *Information*, Vol. 14, No. 5, 2023, p. 261.

[4] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta et al., "When and why your code starts to smell bad," in *International Conference on Software Engineering*, Vol. 1. IEEE, 2015, pp. 403–414.

[5] S.M. Olbrich, D.S. Cruzes, and D.I. Sjøberg, "Are all code smells harmful? A study of god classes and brain classes in the evolution of three open source systems," in *International Conference on Software Maintenance*. IEEE, 2010, pp. 1–10.

[6] M. Fowler, *Refactoring: Improving the design of existing code*. Addison-Wesley Professional, 2018.

[7] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, 2005, pp. 214–223.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann et al., "The WEKA data mining software: An update," *ACM SIGKDD Explorations Newsletter*, Vol. 11, No. 1, 2009, pp. 10–18.

[9] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in *20th International Conference on Software Maintenance*. IEEE, 2004, pp. 350–359.

[10] G. Travassos, F. Shull, M. Fredericks, and V.R. Basili, "Detecting defects in object-oriented designs: using reading techniques to increase software quality," *ACM Sigplan Notices*, Vol. 34, No. 10, 1999, pp. 47–56.
[11] G. Ganea, I. Verebi, and R. Marinescu, "Continuous quality assessment with inCode," *Science of Computer Programming*, Vol. 134, 2017, pp. 19–36.
[12] H. Li and S. Thompson, "Let's make refactoring tools user-extensible!" in *Proceedings of the Fifth Workshop on Refactoring Tools*, 2012, pp. 32–39.
[13] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo, "A review-based comparative study of bad smell detection tools," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016, pp. 1–12.
[14] M.V. Mäntylä and C. Lassenius, "Subjective evaluation of software evolvability using code smells: An empirical study," *Empirical Software Engineering*, Vol. 11, No. 3, 2006, pp. 395–431.
[15] E. Alpaydin, *Introduction to machine learning*. MIT Press, 2020.
[16] S. Jain and A. Saha, "Improving performance by genetically optimizing support vector machine to detect code smells," in *Proceedings of the International Conference on Smart Data Intelligence (ICSMDI 2021)*, 2021.
[17] G.A. Pradipta, R. Wardoyo, A. Musdholifah, I.N.H. Sanjaya, and M. Ismail, "SMOTE for handling imbalanced data problem: A review," in *Sixth International Conference on Informatics and Computing (ICIC)*. IEEE, 2021, pp. 1–8.
[18] H. Gupta, S. Misra, L. Kumar, and N. Murthy, "An empirical study to investigate data sampling techniques for improving code-smell prediction using imbalanced data," in *International Conference on Information and Communication Technology and Applications*. Springer, 2020, pp. 220–233.
[19] S. Jain and A. Saha, "Improving performance with hybrid feature selection and ensemble machine learning techniques for code smell detection," *Science of Computer Programming*, Vol. 212, 2021, p. 102713.
[20] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, Vol. 1. New York: Springer Series in Statistics, 2001.
[21] I. Syarif, A. Prugel-Bennett, and G. Wills, "SVM parameter optimization using grid search and genetic algorithm to improve classification performance," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, Vol. 14, No. 4, 2016, pp. 1502–1509.
[22] D.M. Belete and M.D. Huchaiah, "Grid search in hyperparameter optimization of machine learning models for prediction of hiv/aids test results," *International Journal of Computers and Applications*, Vol. 44, No. 9, 2022, pp. 875–886.
[23] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A.M. Karimi-Mamaghan, and E.G. Talbi, "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art," *European Journal of Operational Research*, Vol. 296, No. 2, 2022, pp. 393–422.
[24] V. Chandra S.S. and H.S. Anand, "Nature inspired meta heuristic algorithms for optimization problems," *Computing*, Vol. 104, No. 2, 2022, pp. 251–269.
[25] E. Osaba, E. Villar-Rodriguez, J. Del Ser, A.J. Nebro, D. Molina et al., "A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems," *Swarm and Evolutionary Computation*, Vol. 64, 2021, p. 100888.
[26] J. McDermott, "When and why metaheuristics researchers can ignore "No Free Lunch" theorems," *SN Computer Science*, Vol. 1, No. 1, 2020, p. 60.
[27] V.W. Porto, "Evolutionary programming," in *Evolutionary Computation*. CRC Press, 2018, pp. 127–140.
[28] A. Lambora, K. Gupta, and K. Chopra, "Genetic algorithm – A literature review," in *International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. IEEE, 2019, pp. 380–384.
[29] M. Pant, H. Zaheer, L. Garcia-Hernandez, A. Abraham et al., "Differential Evolution: A review of more than two decades of research," *Engineering Applications of Artificial Intelligence*, Vol. 90, 2020, p. 103479.

[30] M.G.P. de Lacerda, L.F. de Araujo Pessoa, F.B. de Lima Neto, T.B. Ludermir, and H. Kuchen, "A systematic literature review on general parameter control for evolutionary and swarm-based algorithms," *Swarm and Evolutionary Computation*, Vol. 60, 2021, p. 100777.

[31] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 International Conference on Neural Networks*, Vol. 4. IEEE, 1995, pp. 1942–1948.

[32] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, Vol. 1, No. 4, 2006, pp. 28–39.

[33] S. Chattopadhyay, A. Marik, and R. Pramanik, "A brief overview of physics-inspired metaheuristic optimization techniques," *arXiv preprint arXiv:2201.12810*, 2022.

[34] K. Karthikeyan and P. Dhal, "Multi verse optimization (MVO) technique based voltage stability analysis through continuation power flow in IEEE 57 bus," *Energy Procedia*, Vol. 117, 2017, pp. 583–591.

[35] Z. Wei, C. Huang, X. Wang, T. Han, and Y. Li, "Nuclear reaction optimization: A novel and powerful physics-based algorithm for global optimization," *IEEE Access*, Vol. 7, 2019, pp. 66 084–66 109.

[36] S. Cheng, Q. Qin, J. Chen, and Y. Shi, "Brain storm optimization algorithm: a review," *Artificial Intelligence Review*, Vol. 46, 2016, pp. 445–458.

[37] T. Rahkar Farshi, "Battle royale optimization algorithm," *Neural Computing and Applications*, Vol. 33, No. 4, 2021, pp. 1139–1157.

[38] M.D. Li, H. Zhao, X.W. Weng, and T. Han, "A novel nature-inspired algorithm for optimization: Virus colony search," *Advances in Engineering Software*, Vol. 92, 2016, pp. 65–88.

[39] G.G. Wang, S. Deb, and L.D.S. Coelho, "Earthworm optimisation algorithm: A bio-inspired metaheuristic algorithm for global optimisation problems," *International Journal of Bio-Inspired Computation*, Vol. 12, No. 1, 2018, pp. 1–22.

[40] S. Chinnasamy, M. Ramachandran, M. Amudha, and K. Ramu, "A review on hill climbing optimization methodology," *Recent Trends in Management and Commerce*, Vol. 3, No. 1, 2022.

[41] A.I. Hafez, H.M. Zawbaa, E. Emary, and A.E. Hassanien, "Sine cosine optimization algorithm for feature selection," in *International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*. IEEE, 2016, pp. 1–5.

[42] S. Hassaine, F. Khomh, Y.G. Guéhéneuc, and S. Hamel, "IDS: An immune-inspired approach for the detection of software design smells," in *Seventh International Conference on the Quality of Information and Communications Technology*. IEEE, 2010, pp. 343–348.

[43] N. Moha, Y.G. Guéhéneuc, L. Duchien, and A.F. Le Meur, "Decor: A method for the specification and detection of code and design smells," *IEEE Transactions on Software Engineering*, Vol. 36, No. 1, 2009, pp. 20–36.

[44] A. Maiga, N. Ali, N. Bhattacharya, A. Sabane, Y.G. Guéhéneuc et al., "Smurf: A SVM-based incremental anti-pattern detection approach," in *19th Working conference on reverse engineering*. IEEE, 2012, pp. 466–475.

[45] F. Khomh, S. Vaucher, Y.G. Guéhéneuc, and H. Sahraoui, "BDTEX: A GQM-based Bayesian approach for the detection of antipatterns," *Journal of Systems and Software*, Vol. 84, No. 4, 2011, pp. 559–572.

[46] F.A. Fontana, M.V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Software Engineering*, Vol. 21, No. 3, 2016, pp. 1143–1191.

[47] M. Kessentini and A. Ouni, "Detecting android smells using multi-objective genetic programming," in *International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2017, pp. 122–132.

[48] A. Kaur, S. Jain, and S. Goel, "SP-J48: A novel optimization and machine-learning-based approach for solving complex problems: special application in software engineering for detecting code smells," *Neural Computing and Applications*, Vol. 32, No. 11, 2020, pp. 7009–7027.

[49] S. Jain and A. Saha, "Rank-based univariate feature selection methods on machine learning classifiers for code smell detection," *Evolutionary Intelligence*, Vol. 15, No. 1, 2022, pp. 609–638.

[50] M. Boussaa, W. Kessentini, M. Kessentini, S. Bechikh, and S. Ben Chikha, "Competitive coevolutionary code-smells detection," in *Search Based Software Engineering: 5th International Symposium, SSBSE 2013, St. Petersburg, Russia, August 24–26, 2013. Proceedings 5*. Springer, 2013, pp. 50–65.

[51] W. Kessentini, M. Kessentini, H. Sahraoui, S. Bechikh, and A. Ouni, "A cooperative parallel search-based software engineering approach for code-smells detection," *IEEE Transactions on Software Engineering*, Vol. 40, No. 9, 2014, pp. 841–861.

[52] D. Sahin, M. Kessentini, S. Bechikh, and K. Deb, "Code-smell detection as a bilevel problem," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 24, No. 1, 2014, pp. 1–44.

[53] U. Mansoor, M. Kessentini, B.R. Maxim, and K. Deb, "Multi-objective code-smells detection using good and bad design examples," *Software Quality Journal*, Vol. 25, No. 2, 2017, pp. 529–552.

[54] G. Saranya, H.K. Nehemiah, A. Kannan, and V. Nithya, "Model level code smell detection using egapso based on similarity measures," *Alexandria Engineering Journal*, Vol. 57, No. 3, 2018, pp. 1631–1642.

[55] G. Saranya, H.K. Nehemiah, and A. Kannan, "Hybrid particle swarm optimisation with mutation for code smell detection," *International Journal of Bio-Inspired Computation*, Vol. 12, No. 3, 2018, pp. 186–195.

[56] M.M. Draz, M.S. Farhan, S.N. Abdulkader, and M. Gafar, "Code smell detection using whale optimization algorithm," *CMC-Computers Materials and Continua*, Vol. 68, No. 2, 2021, pp. 1919–1935.

[57] B. Amal, M. Kessentini, S. Bechikh, J. Dea, and L.B. Said, "On the use of machine learning and search-based software engineering for ill-defined fitness function: a case study on software refactoring," in *International Symposium on Search Based Software Engineering*. Springer, 2014, pp. 31–45.

[58] A. Ghannem, G.E. Boussaidi, and M. Kessentini, "Model refactoring using interactive genetic algorithm," in *International Symposium on Search Based Software Engineering*. Springer, 2013, pp. 96–110.

[59] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "JDeodorant: identification and application of extract class refactorings," in *33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 1037–1039.

[60] T.J. Dea, M. Kessentini, W.I. Grosky, and K. Deb, "Software refactoring using cooperative parallel evolutionary algorithms," 2016.

[61] G. Saranya, H. Nehemiah, A. Kannan, and V. Pavithra, "Prioritizing code smell correction task using strength pareto evolutionary algorithm," *Indian Journal of Science and Technology*, Vol. 11, No. 20, 2018, pp. 1–12.

[62] A. Ouni, M. Kessentini, S. Bechikh, and H. Sahraoui, "Prioritizing code-smells correction tasks using chemical reaction optimization," *Software Quality Journal*, Vol. 23, No. 2, 2015, pp. 323–361.

[63] A. Kaur, S. Jain, and S. Goel, "Sandpiper optimization algorithm: a novel approach for solving real-life engineering problems," *Applied Intelligence*, Vol. 50, No. 2, 2020, pp. 582–619.

[64] G. Lacerda, F. Petrillo, M. Pimenta, and Y.G. Guéhéneuc, "Code smells and refactoring: A tertiary systematic review of challenges and observations," *Journal of Systems and Software*, Vol. 167, 2020, p. 110610.

[65] R.S. Menshawy, A.H. Yousef, and A. Salem, "Code smells and detection techniques: A survey," in *International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*. IEEE, 2021, pp. 78–83.

[66] M. Zhang, T. Hall, and N. Baddoo, "Code bad smells: A review of current knowledge," *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 23, No. 3, 2011, pp. 179–202.

[67] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li et al., "The Qualitas Corpus: A curated collection of Java code for empirical studies," in *Asia Pacific Software Engineering Conference*. IEEE, 2010, pp. 336–345.

[68] G. Van Rossum and F.L. Drake, Jr., *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.
[69] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, Vol. 12, 2011, pp. 2825–2830.
[70] S.B. Kotsiantis, D. Kanellopoulos, and P.E. Pintelas, "Data preprocessing for supervised leaning," *International Journal of Computer Science*, Vol. 1, No. 2, 2006, pp. 111–117.
[71] C.V.G. Zelaya, "Towards explaining the effects of data preprocessing on machine learning," in *35th international conference on data engineering (ICDE)*. IEEE, 2019, pp. 2086–2090.
[72] M. Mehmood, N. Alshammari, S.A. Alanazi, and F. Ahmad, "Systematic framework to predict early-stage liver carcinoma using hybrid of feature selection techniques and regression techniques," *Complexity*, Vol. 2022, 2022, pp. 1–11.
[73] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise Reduction in Speech Processing*. Springer, 2009, pp. 1–4.
[74] T.T. Wong and P.Y. Yeh, "Reliable accuracy estimates from k-fold cross validation," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 32, No. 8, 2019, pp. 1586–1594.
[75] L. Abualigah, A. Diabat, S. Mirjalili, M. Abd Elaziz, and A.H. Gandomi, "The arithmetic optimization algorithm," *Computer Methods in Applied Mechanics and Engineering*, Vol. 376, 2021, p. 113609.
[76] J.S. Chou and D.N. Truong, "A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean," *Applied Mathematics and Computation*, Vol. 389, 2021, p. 125535.
[77] H. Karami, M.V. Anaraki, S. Farzin, and S. Mirjalili, "Flow direction algorithm (FDA): A novel optimization approach for solving optimization problems," *Computers and Industrial Engineering*, Vol. 156, 2021, p. 107224.
[78] B. Das, V. Mukherjee, and D. Das, "Student psychology based optimization algorithm: A new population based optimization algorithm for solving optimization problems," *Advances in Engineering software*, Vol. 146, 2020, p. 102804.
[79] H. Yapici and N. Cetinkaya, "A new meta-heuristic optimizer: Pathfinder algorithm," *Applied Soft Computing*, Vol. 78, 2019, pp. 545–568.
[80] S. Mirjalili, "SCA: A sine cosine algorithm for solving optimization problems," *Knowledge-Based Systems*, Vol. 96, 2016, pp. 120–133.
[81] R. Rao, "Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems," *International Journal of Industrial Engineering Computations*, Vol. 7, No. 1, 2016, pp. 19–34.
[82] A. Askarzadeh, "A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm," *Computers and Structures*, Vol. 169, 2016, pp. 1–12.
[83] S. Mirjalili, "Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Computing and Applications*, Vol. 27, 2016, pp. 1053–1073.
[84] A.H. Gandomi and A.H. Alavi, "Krill herd: A new bio-inspired optimization algorithm," *Communications in Nonlinear Science and Numerical Simulation*, Vol. 17, No. 12, 2012, pp. 4831–4845.
[85] S. Mirjalili, S.M. Mirjalili, and A. Hatamlou, "Multi-verse optimizer: a nature-inspired algorithm for global optimization," *Neural Computing and Applications*, Vol. 27, No. 2, 2016, pp. 495–513.
[86] M.Y. Cheng and D. Prayogo, "Symbiotic organisms search: A new metaheuristic optimization algorithm," *Computers and Structures*, Vol. 139, 2014, pp. 98–112.
[87] X.S. Yang, "Flower pollination algorithm for global optimization," in *International Conference on Unconventional Computing and Natural Computation*. Springer, 2012, pp. 240–249.
[88] R.V. Rao, V.J. Savsani, and D. Vakharia, "Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems," *Computer-Aided Design*, Vol. 43, No. 3, 2011, pp. 303–315.
[89] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "GSA: A gravitational search algorithm," *Information Sciences*, Vol. 179, No. 13, 2009, pp. 2232–2248.

[90] D. Simon, "Biogeography-based optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 12, No. 6, 2008, pp. 702–713.
[91] W.S. Noble, "What is a support vector machine?" *Nature Biotechnology*, Vol. 24, No. 12, 2006, pp. 1565–1567.
[92] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "KNN model-based approach in classification," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences*. Springer, 2003, pp. 986–996.
[93] M.N. Ab Wahab, S. Nefti-Meziani, and A. Atyabi, "A comprehensive review of swarm optimization algorithms," *PloS One*, Vol. 10, No. 5, 2015, p. e0122827.
[94] T.A. Jumani, M.W. Mustafa, A.S. Alghamdi, M.M. Rasid, A. Alamgir et al., "Swarm intelligence-based optimization techniques for dynamic response and power quality enhancement of AC microgrids: A comprehensive review," *IEEE Access*, Vol. 8, 2020, pp. 75 986–76 001.
[95] V. López, A. Fernández, and F. Herrera, "On the importance of the validation technique for classification with imbalanced datasets: Addressing covariate shift when data is skewed," *Information Sciences*, Vol. 257, 2014, pp. 1–13.
[96] R.F. Woolson, "Wilcoxon signed-rank test," *Wiley Encyclopedia of Clinical Trials*, 2007, pp. 1–3.
[97] T. Harris and J.W. Hardin, "Exact Wilcoxon signed-rank and Wilcoxon Mann–Whitney ranksum tests," *The Stata Journal*, Vol. 13, No. 2, 2013, pp. 337–343.
[98] D.H. Wolpert and W.G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, 1997, pp. 67–82.
[99] M. Meselhi, R. Sarker, D. Essam, and S. Elsayed, "A decomposition approach for large-scale non-separable optimization problems," *Applied Soft Computing*, Vol. 115, 2022, p. 108168.
[100] Z. Yang, H. Qiu, L. Gao, D. Xu, and Y. Liu, "A general framework of surrogate-assisted evolutionary algorithms for solving computationally expensive constrained optimization problems," *Information Sciences*, Vol. 619, 2023, pp. 491–508.
[101] W. Li, T. Zhang, R. Wang, S. Huang, and J. Liang, "Multimodal multi-objective optimization: Comparative study of the state-of-the-art," *Swarm and Evolutionary Computation*, 2023, p. 101253.
[102] Z. Beheshti and S.M.H. Shamsuddin, "A review of population-based meta-heuristic algorithms," *International Journal of Advances in Soft Computing and its Applic*, Vol. 5, No. 1, 2013, pp. 1–35.
[103] H. Grodzicka, A. Ziobrowski, Z. Łakomiak, M. Kawa, and L. Madeyski, "Code smell prediction employing machine learning meets emerging Java language constructs," *Data-Centric Business and Applications: Towards Software Development*, Vol. 4, 2020, pp. 137–167.
[104] S. Suthaharan, "Support vector machine," in *Machine learning models and algorithms for big data classification*. Springer, 2016, pp. 207–235.
[105] O. Kramer, "k-nearest neighbors," in *Dimensionality reduction with unsupervised nearest neighbors*. Springer, 2013, pp. 13–23.
[106] J. Han, J. Pei, and M. Kamber, *Data mining: Concepts and techniques*. Elsevier, 2011.
[107] J.F. O'Callaghan and D.M. Mark, "The extraction of drainage networks from digital elevation data," *Computer Vision, Graphics, and Image Processing*, Vol. 28, No. 3, 1984, pp. 323–344.

Appendix A. Machine learning algorithms used

Code smell detection exploits classification methodology as output is binary, that is, if the class/method is affected by a particular smell or not. We have implemented SVM and k-NN binary classifiers.

A.1. Support Vector Machine (SVM)

SVM is one of the most versatile machine learning algorithms for classification and regression problems. It assumes that two adjacent instances in input space must have the same output value [91]. It calculates an optimal hyperplane that separates classes in multi-dimensional space. The class of a new instance is marked by its position on which side of the hyperplane. Let's take N training samples $A = \{a_1, a_2, \ldots, a_N\}$ where a_i means i-th training sample and has f features. It is associated with one of the two classes $b_i \in \{0, 1\}$. So, complete training set is $\{(a_1, b_1), (a_2, b_2), \ldots, (a_N, b_N)\}$ where a_i training sample belongs to class b_i. The equation of hyperplane to be optimized is:

$$L_P = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\epsilon_i - \sum_{i=1}^{N}\alpha_i\left(b_i(w^T a_i + y) - 1 + \epsilon_i\right) \tag{A.1.1}$$

where w is the weight vector, normal to the hyperplane, y is the threshold, and a is the input data point in d dimensional space. The algorithm aims to select the best values for threshold and weight such that the hyperplane is as far as possible from the closest data points. ϵ_i is the slack variable greater than equal to 0, and it is to be minimised. Each ϵ_i represents the distance between the i-th data point and the corresponding margin hyperplane. C is the regularization parameter that controls the trade-off between the slack variable penalty and the size of the margin. $\alpha_i \geq 0$ and $i = 1, 2, \ldots, N$. α_i are the Lagrange multipliers, and each α_i corresponds to one data point (a_i, b_i) and L_P becomes the primal equation that is to be optimized [104].

A.2. k-Nearest Neighbors

k-NN is one of the most popular machine learning algorithm and is known for its simplicity. It can be executed for both classification and regression problems. It analyses the parametric estimation of unknown probabilities, which are otherwise difficult to predict. The main idea behind this algorithm is that the class of a new data point is decided by its majority of k-neighbors. k-NN considers k nearest instances to determine the class of query instance and selects one with the highest frequency [105]. The distance metric is used to calculate the relative distance between instances in n-dimensional space, where n is the number of features. Minkowski distance is calculated as:

$$\sqrt[p]{\sum_{i=1}^{n}|x_i - y_i|^p}$$

It is a generalized distance metric. For Manhattan distance, p is equal to 1. For Euclidean distance, p is equal to 0, and for Chebyshev distance, $p \to \infty$ [106]. Meta-heuristic algorithms are used to retrieve the best values for k and p.

Appendix B. Meta-heuristic algorithms used

Meta-heuristic algorithms are general-purpose algorithms that do not rely on specific problem structures but search the solution space using heuristic rules inspired by natural

phenomena, such as evolution, swarm intelligence, or physics. The current study uses the following meta-heuristic techniques to optimize machine learning algorithms.

B.1. Arithmetic optimization [75]

B.1.1. Behavior

Basic math operations like Addition (+), Subtraction (−), Multiplication (·) and Division (÷).

B.1.2. Phase changing variable

The algorithm strikes a balance between the exploration and exploitation phases using the MOA (Math Optimizer Accelerated) operator, which is calculated using the formula below:

$$MOA(C_t) = \text{Min} + C_t \left(\frac{\text{Max} - \text{Min}}{M_t} \right) \tag{B.1.1}$$

where C_t is t-th or current iteration and its value is between 1 and the maximum number of iterations, M_t, Max and Min are an accelerated function's maximum and minimum values, respectively. If the random number $r1$ is less than MOA, the exploration phase begins; otherwise, the algorithm proceeds with the exploitation phase.

B.1.3. Exploration equations

Exploration is done using either a division or multiplication search strategy due to their high dispersion property. The next position in exploration phase is determined using the following equations and conditions:

$$MOP(C_t) = 1 - \frac{{C_t}^{1/\alpha}}{{M_t}^{1/\alpha}} \tag{B.1.2}$$

$$x_{i,j}(C_t + 1) = \begin{cases} \text{best}(x_j)/(MOP + \text{esp})((UB_j - LB_j)\mu + LB_j) & \text{if } r2 < 0.5 \\ \text{best}(x_j) \cdot MOP((UB_j - LB_j)\mu + LB_j) & \text{otherwise} \end{cases} \tag{B.1.3}$$

where MOP means Math Optimizer Probability, $r2$ = random number that decides operator, C_t = current iteration, M_t = maximum iteration, $MOP(C_t)$ = value of the coefficient at t-th iteration, $\alpha = 5$ is sensitive parameter that defines the exploitation accuracy over the iterations, $x_i(C_t + 1)$ = i-th solution of the next iteration, $x_{i,j}(C_t)$ is i-th solution in the j-th position for the current iteration, best(x_j) is best-obtained solution so far in the j-th position, esp = small integer number, LB_j and UB_j are the lower bound and upper bound of the j-th position, respectively, $\mu = 0.5$ is a controlling parameter that adjusts the search process.

B.1.4. Exploitation equations

Subtraction and addition operators can quickly converge to a result in specific number of iterations due to their low dispersion property, thus perfect for exploitation.The random

number $r3$ is employed to select operator. The next position in the exploitation phase is determined using the following equations and conditions:

$$x_{i,j}(C_t+1) = \begin{cases} \text{best}(x_j) - MOP((UB_j - LB_j)\mu + LB_j) & \text{if } r3 < 0.5 \\ \text{best}(x_j) + MOP((UB_j - LB_j)\mu + LB_j) & \text{otherwise} \end{cases} \tag{B.1.4}$$

Exploitation search operators, addition and subtraction, try to avoid plunging in local optima, and the stochastic nature of μ always allows exploration till the last iteration, introducing diversity in solutions.

B.2. Jellyfish search optimization [76]

B.2.1. Behavior

Swarming behaviour of jellyfish for foraging purposes. The solution is represented by area and its corresponding objective function depicting the quantity of food in that location.

B.2.2. Initialization

The population is initialized using a logistic map, $X_{i+1} = \rho X_i(1 - X_i)$ where $0 \leq X_0 \leq 1$. X_i is the chaotic logistic value of the i-th jellyfish's location and X_0 is a randomly generated location. $X_0, X_i \in [0, 1]$, $X_0 \notin \{0.0, 0.25, 0.5, 0.75, 1.0\}$, and $\rho = 4$. If a jellyfish exceeds the boundaries of the search space, it will be relocated within the boundaries using the following equation:

$$X'_{i,d} = \begin{cases} (X_{i,d} - U_{b,d}) + L_b(d) & \text{if } X_{i,d} > U_{bd} \\ (X_{i,d} - L_{b,d}) + U_b(d) & \text{if } X_{i,d} < L_{bd} \end{cases} \tag{B.2.1}$$

where $X_{i,d}$ = location of the i-th jellyfish in d-dimensional search space, $X'_{i,d}$ = updated location after checking boundary constraints, L_{bd} and U_{bd} are the lower and upper bounds of search space. Jellyfish are attracted to places that have more nutrients or food. The new updated position of jellyfish can be evaluated using the following equations:

$$X_i(t+1) = X_i(t) + r_1 \cdot \overrightarrow{trend} \tag{B.2.2}$$

$$\overrightarrow{trend} = X^* - \beta \cdot r_2 \cdot \mu \tag{B.2.3}$$

where $(\beta > 0)$ = distribution coefficient related to the length of the $\overrightarrow{trend}$, $\overrightarrow{trend}$ = direction of ocean currents evaluated by averaging all the vectors from each jellyfish in the ocean to the jellyfish currently in the best location, X^* = jellyfish currently with the best location in the swarm, $e_c = \beta \cdot r_2$ is the attraction factor, μ = mean location of all the jellyfishes, t = current iteration, r_1, r_2 = random numbers between $[0, 1]$.

B.2.3. Exploration equations

The new position in passive motion is determined using the following equation:

$$X_i(t+1) = X_i(t) + \eta \cdot rand(0,1) \cdot (U_b - L_b) \tag{B.2.4}$$

where L_b and U_b are the lower and upper bounds of the search space, η = motion coefficient, related to the length of motion around each jellyfish's location.

B.2.4. Exploitation equations

The active motion of i-th jellyfish is determined by the relative position of randomly selected j-th jellyfish. The following equations depict their movements:

$$X_i(t+1) = X_i(t) + r_3 \cdot \overrightarrow{Direction} \tag{B.2.5}$$

$$\overrightarrow{Direction} = \begin{cases} X_i(t) - X_j(t) & \text{if } f(X_i) < f(X_j) \\ X_j(t) - X_i(t) & \text{if } f(X_i) \geq f(X_j) \end{cases} \tag{B.2.6}$$

where $f(X_i)$, $f(X_j)$ = objective function values of i-th and j-th jellyfish, respectively, $\overrightarrow{Direction}$ = vector of the active motion, r_3 = random number between $[0, 1]$, t = current iteration.

B.2.5. Phase changing variable

Time control function $c(t)$ is employed to decide the active and passive motion of jellyfish inside the bloom and also their movements toward ocean currents. It is calculated using the following formulae:

$$c(t) = \left| \left(1 - \frac{t}{t_{\max}}\right) \cdot (2 \cdot r_4 - 1) \right| \tag{B.2.7}$$

where t = current iteration, $t_{\max}$ = maximum iterations, $r_4 \in [0, 1]$ = random number. The control function $c(t)$ fluctuates between 0 and 1 and decreases as iteration progresses. If the value of $(c(t) > C_0)$, the jellyfish follows the ocean current, otherwise, the jellyfish moves inside the jellyfish bloom. To determine the movement of jellyfish inside the swarm, the function $[1 - c(t)]$ is employed. If $(r_4 > [1 - c(t)])$, passive motion is favored otherwise, active motion is favored. That is how the algorithm converges to find an optimal solution and stops when the end criteria are met.

B.3. Flow direction optimization [77]

B.3.1. Behavior

The drainage basin system.

B.3.2. Initialization

It uses the D8 algorithm [107] to determine the flow direction of direct runoff (amount of water remaining on the ground surface after precipitation and mislaying such as interception, evapotranspiration, and infiltration). Direct runoff can be calculated using the formula:

$$r_d = \sum_{m=1}^{M} (R_m - \phi\delta t) \tag{B.3.1}$$

where ϕ = average amount of water loss during rainfall, R_m = rainfall, δt = time interval, M = total number of time steps.

Flows are population in drainage basin/search space with height as its objective function. Each flow, flows in a direction towards the lowest altitude with velocity V. The most optimal objective function is the basin's outlet point. Each flow with β neighbors has a neighborhood radius of δ and total population members are α. The initial position of flow is calculated using the following formula:

$$Flow\_X(i) = lb + rand \cdot (ub - lb) \tag{B.3.2}$$

where ub and lb are upper and lower limits of the decision variables and $rand$ is a random number between $[0, 1]$ with uniform distribution. The position of the neighboring j-th flow can be determined using the following relation:

$$Nghbr\_X(j) = Flow\_X(i) + rand_n \cdot \delta \tag{B.3.3}$$

where $rand_n$ is a random value with a normal distribution, a mean of zero, and a standard deviation of 1.

B.3.3. Phase changing variable

δ determines the phase of the algorithm. The small value of δ means exploitation, and the large value means exploration. δ starts with a significant value and is reduced over the iterations to support finding global solution and avoiding trapping in local optima. Randomness is introduced for that.

$$\delta = (rand \cdot Xrand - rand \cdot Flow\_X(i)) \cdot ||Best\_X - Flow\_X(i)|| \cdot W \tag{B.3.4}$$

where $rand$ = random number, $Xrand$ = random position calculated using (B.3.1), W = non-linear weight with a random number from zero to infinity and is calculated using the following relation:

$$W = \left(\left(1 - \frac{iter}{\mathrm{Max}_{iter}}\right)^{2 \cdot rand_n}\right) \cdot \left(\overline{rand} \cdot \frac{iter}{\mathrm{Max}_{iter}}\right) \cdot \overline{rand} \tag{B.3.5}$$

where $\overline{rand}$ is a random vector with uniform distribution.

B.3.4. Learning equations

Over the iteration, W has large variation, $Flow\_X(i)$ is closer to $Best\_X$, and the Euclidian distance between $Best\_X$ and $Flow\_X(i)$ is reduced to zero, bringing us closer to optimal solution. Each flow with V velocity moves towards the best neighbor. If the best neighbor has a better fitness value than that of a current flow, the flow velocity vector is updated using formula:

$$V = rand_n \cdot S_0 \tag{B.3.6}$$

where S_0 is the slope vector between the neighbor and the current position of the flow. Random number $rand_n$ reinforces exploration. The slope between neighbors i and j can be evaluated using the following calculation:

$$S_0(i,j,d) = \frac{Flow\_\text{fit}(i) - Nghbr\_\text{fit}(j)}{||Flow\_x(i,d) - Nghbr\_x(j,d)||} \tag{B.3.7}$$

where d is dimension of the problem. The new position can be calculated using following equation:

$$NewF_X(i) = Flow\_X(i) + V \cdot \frac{Flow\_X(i) - Nghbr\_X(j)}{||Flow\_x(i) - Nghbr\_x(j)||} \tag{B.3.8}$$

It is also possible that the fitness function of all neighbors is not less than the current flow, and then the algorithm randomly chooses another flow. The following relation shows how to simulate the flow direction under these conditions:

$$NewF\_X(i) = \begin{cases} Flow\_X(i) + rand_n \cdot (Flow\_X(r) - Flow\_X(i)) \\ \qquad\qquad \text{if } Flow\_\text{fit}(r) < Flow\_\text{fit}(i) \\ Flow\_X(i) + 2rand_n \cdot (Best\_X - Flow\_X(i)) \\ \qquad\qquad \text{otherwise} \end{cases} \tag{B.3.9}$$

where r and $rand_n$ are random integers.

B.4. Student psychology Based Optimization [78]

B.4.1. Behavior

The psychology of students making genuine efforts to improve their marks.

B.4.2. Learning equations

Students' overall marks are enhanced if the marks in each subject they are offered improves. Depending on their interest in a subject, the student may give more effort to improve overall marks. Students are categorized into four types based on their psychology:

(i) **Best Student.** The student who has the maximum overall marks is said to be the best student. They will try to maintain their position by putting in more effort than any randomly chosen student. Improvement of the best student can be evaluated using following equation:

$$X_{\text{bestnew}} = X_{\text{best}} + (-1)^k \cdot rand \cdot (X_{\text{best}} - X_j) \tag{B.4.1}$$

where X_{best} = marks obtained by best student, X_j = marks of randomly selected j-th student, $rand \in [0, 1]$ = random number, $k = 1$ or 2.

(ii) **Good Student.** The student who will try to give more effort in the subject of their interest to improve overall performance is a good student. They are subject-wise good students and are random because psychologies differ for each student.

$$X_{\text{new } i} = X_{\text{best}} + rand \cdot (X_{\text{best}} - X_i) \tag{B.4.2}$$

$$X_{\text{new } i} = X_i + rand \cdot (X_{\text{best}} - X_i) + rand \cdot (X_i - X_{\text{mean}}) \tag{B.4.3}$$

where X_i = marks of i-th student in that subject, X_{mean} = average marks of the class in that subject, If the student tries to give more or a similar effort to that of the best student, its improvement can be calculated using Eq (B.4.2). If the student gives more effort than an average student and the effort provided by the best student, their marks can be evaluated using Eq (B.4.3).

(iii) **Average Student.** While giving average effort to the subject, they will provide more effort to other exciting subjects. Their improvement can be calculated using the below formulae:

$$X_{\text{new } i} = X_i + rand \cdot (X_{\text{mean}} - X_i) \tag{B.4.4}$$

(iv) **Students trying randomly to improve.** They give random efforts to the subject irrespective of the students mentioned above. Their performance can be evaluated using the following formulae:

$$X_{\text{new } i} = X_{\min} + \ rand \cdot (X_{\max} - X_{\min}) \tag{B.4.5}$$

where $X_{\max}$ and $X_{\min}$ are the upper and lower bound on marks of the subject, respectively.

B.5. Pathfinder optimization [79]

B.5.1. Behavior

Swarms for foraging, breeding, and hunting purposes.

B.5.2. Learning equations

Each individual is a candidate solution in a d-dimensional space having a position vector as the fitness function. The algorithm is modeled to find prey as follows:

$$x_i^{t+1} = x_i^t + R_1 \cdot (x_j^t - x_i^t) + R_2 \cdot (x_p^t - x_i^t) + \eta, \quad i \geq 2 \tag{B.5.1}$$

where t = current iteration, x_i = position vector of the i-th search agent, x_j = position vector of the j-th search agent, x_p = position of pathfinder (leader), $R_1 = \alpha r_1$ and $R_2 = \beta r_2$, $r_1, r_2 \in [0, 1]$ = random numbers, α and β are randomly selected in the range of $[1, 2]$, α = coefficient of interaction that decides the magnitude of movement between two neighbors, β = coefficient of attraction that decides the movement of the herd with the leader, η is the vibration, which can be calculated using the following formulae:

$$\eta = \left(1 - \frac{t}{t_{\max}}\right) \cdot u_1 \cdot D_{ij}, \ \ D_{ij} = ||x_i - x_j|| \tag{B.5.2}$$

where $u_1 \in [-1, 1]$ = random vector, D_{ij} = distance between two members, $t_{\max}$ = maximum iteration. The position of the pathfinder is updated according to the following equation:

$$x_p^{t+1} = x_p^t + 2r_3 \cdot (x_p^t - x_p^{t-1}) + A \tag{B.5.3}$$

where u_2 and r_3 are random vectors between $[-1, 1]$ and $[0, 1]$, respectively.

B.5.3. Phase Changing Variable

A is fluctuation factor, responsible for switching in the exploration and exploitation phases. It is calculated as follows:

$$A = u_2 \cdot e^{\frac{-2t}{t_{\max}}} \tag{B.5.4}$$

The position of the pathfinder provides the global optimum solution and converges with increasing iterations.

B.6. Sine Cosine Optimization [80]

B.6.1. Behavior

Sine and cosine functions.

B.6.2. Learning equations

The following equation decides the improved position and phase in Sine Cosine Optimization:

$$X_i^{t+1} = \begin{cases} X_i^t + r_1 \cdot \sin(r_2) \cdot |r_3 P_i^t - X_i^t|, & r_4 < 0.5 \\ X_i^t + r_1 \cdot \cos(r_2) \cdot |r_3 P_i^t - X_{ii}^t|, & r_4 \geq 0.5 \end{cases} \tag{B.6.1}$$

and

$$r_1 = a - t\frac{a}{T} \tag{B.6.2}$$

where X_i^t = position of the current solution in the i-th dimension at the t-th iteration, P_i^t = position of the destination point in the i-th dimension, r_1, r_2, r_3 and r_4 are random numbers, The r_1 is adaptive change calculated using Eq. (B.6.2), which is responsible for selecting the next search area; higher the value of r_1, the greater is the search area, t = current iteration, T = maximum number of iterations, a is a constant value, r_2 parameter decides the extent of the movement towards or away from the target and is in range $[0, 2\pi]$. r_3 is in the range $[-2, 2]$ and is a random weight score for the target that randomly asserts $(r3 > 1)$ or refutes $(r3 < 1)$ the influence of the target in determining the distance. r_4 is used to switch between the sine and cosine functions and lies between $[0, 1]$. The algorithm explores the search space when the sine and cosine functions range in $(1, 2]$ and $[-2, -1)$. However, exploits when the range is in the interval of $[-1, 1]$.

B.7. Jaya Optimization [81]

B.7.1. Behavior

The value of function $f(x)$, trying to get closer to $f(x)_{\text{best}}$, the best value and avoid $f(x)_{\text{worst}}$, the worst value of function to be optimized.

B.7.2. Learning equations

If n is the number of candidate solutions ($k = 1, 2, \ldots, n$) and m is number of design variables ($j = 1, 2, \ldots, m$), then $X_{j,k,i}$ is the value of j-th variable for the k-th candidate during i-th iteration. It is updated based on the following equation:

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,i}(X_{j,b,i} - |X_{j,k,i}|) - r_{2,j,i}(X_{j,w,i} - |X_{j,k,i}|) \tag{B.7.1}$$

where $X_{j,b,i}$ and $X_{j,w,i}$ is the best and worst value for j-th variable. $r_{1,j,i}$ and $r_{2,j,i}$ is the random number between $[0, 1]$. If updated solution is better, it is accepted and becomes input for next iteration.

B.8. Crow Search Optimization[82]

B.8.1. Behavior

Crow's mindful and intelligent behavior of stealing and hiding food.

B.8.2. Learning equations

Crows live in flocks and remember the hiding place of their food. They follow each other to steal the food and change their hiding places to avoid theft using probability. To begin with, positions of N crows are randomly initialized in d-dimensional search space. With iteration t, crow i will have memory of its hiding place, $m_{i,t}$. This is the best position that crow i has obtained so far.

Crows follow each other to search for the other food sources. Two cases arises, that is, if crow knows it is being followed or not. If crow j doesn't know that it is being followed by crow i, crow i will change its position according to first case. If crow j knows it is being followed by crow i, then random position is assigned. Following are the modeled equations:

$$x_{i,t+1} = \begin{cases} x_{i,t} + r_i \cdot fl_{i,t} \cdot (m_{j,t} - x_{i,t}) & r_i \geq AP_{j,t} \\ \text{random position} & \text{otherwise} \end{cases} \tag{B.8.1}$$

where $r_i \in [0, 1]$ is a random number and $fl_{i,t}$ is the flight length of i-th crow at t-th iteration. If value of fl is small, local search is favored otherwise global search is supported. $AP_{j,t}$ and $m_{j,t}$ is the awareness probability and memory of j-th crow at iteration t, respectively. AP helps in switching phases as high value of AP helps in exploration, while small value of AP guides toward exploitation. Memory function is updated as follows:

$$m_{i,t+1} = \begin{cases} F(x_{i,t+1}) & F(x_{i,t+1}) < F(m_{i,t}) \\ m_{i,k} & \text{otherwise} \end{cases} \tag{B.8.2}$$

where $F(\cdot)$ represents the objective function.

B.9. Dragonfly optimization [83]

B.9.1. Behavior

Swarming behavior of dragonflies – hunting (static swarm) and migration (dynamic swarm).

B.9.2. Learning equations

Position of each search agent is updated with two vectors, step (ΔX) and position (X). Step vector represents direction of the movement of dragonflies and is calculated by adding all the properties:

$$\Delta X_{t+1} = sS_i + aA_i + cC_i + fF_i + eE_i + w\Delta X_t \tag{B.9.1}$$

where s, a, c, f, and e are the weights associated with Separation $\left(S_i = -\sum_{j=1}^{N}(X - X_j)\right)$, Alignment $\left(A_i = \sum_{j=1}^{N} V_j \Big/ N\right)$, Cohesion $\left(C_i = \sum_{j=1}^{N} X_j \Big/ N - X\right)$, Attraction $(F_i = X^+ - X)$, and Distraction $(E_i = X^- + X)$ of i-th search agent, N = total number of neighboring agents, X = current agent, X_j = position of the j-th neighbor, V_j = velocity of the j-th neighbor, X^+ = position of the food, X^- = position of the enemy, w = inertia weight, t = iteration count. Position vector, $X_{t+1} = X_t + \Delta X_{t+1}$ is calculated next.

Swarming weights (s, a, c, f, e, and w) are tuned adaptively and the radii of neighborhoods are increased proportional to the number of iterations to strike the balance between exploration and exploitation. To add to the randomness and exploration of the dragonflies movement, Lévy flight is being introduced in the new position as follows: $X_{t+1} = X_t + LF \cdot X_t$.

B.10. Krill herd optimization [84]

B.10.1. Behavior

Swarming and foraging behaviour of krills.

B.10.2. Learning equations

The position of a krill is dependent on three crucial factors – movement induced by other krills (N_i), foraging motion (F_i), and random diffusion (D_i). This is modeled by the following equation:

$$\frac{dx_i}{dt} = N_i + F_i + D_i \tag{B.10.1}$$

(i) **Motion induced by other krills.** The direction of motion induced for the i-th krill is:

$$N_i^{\text{new}} = N_{\max}\alpha_i + \omega_n N_i^{\text{old}} \tag{B.10.2}$$

where $N_{\max}$ = maximum speed induced, ω_n = inertia weight of motion induced in the range $[0, 1]$, N_i^{old} = last motion induced, $\alpha_i = \alpha_i^{\text{local}} + \alpha_i^{\text{target}}$, α_i^{local} = local effect provided by the neighbors, α_i^{target} = target direction effect provided by the best krill. The impact of the neighboring krills in a krill movement is evaluated as follows:

$$\alpha_i^{\text{local}} = \sum_{j=1}^{n} \hat{K}_{i,j}\hat{x}_{i,j} \tag{B.10.3}$$

where $\hat{K}_{i,j}$ = normalized value of the similarity vector of the i-th krill, $\hat{x}_{i,j}$ = normalized value of related positions for the i-th krill, n = total number of neighboring krills. $\hat{K}_{i,j}$ is evaluated as:

$$\hat{K}_{i,j} = \frac{K_i - K_j}{K^{\text{worst}} - K^{\text{best}}} \tag{B.10.4}$$

where K_i and K_j are the fitness value of i-th and j-th neighboring krill, K^{worst} and K^{best} are the worst and best value of fitness for a krill so far, respectively, $\hat{x}_{i,j}$ is evaluated as:

$$\hat{x}_{i,j} = \frac{x_j - x_i}{||x_j - x_i|| + \epsilon} \tag{B.10.5}$$

where x_i and x_j are the positions of i-th krill and neighboring j-th krill, $||x_j - x_i||$ is the distance between j-th and i-th krill, ϵ is a small positive number added to avoid singularities. A sensing distance (d_s) is evaluated for each krill using formulae:

$$d_{si} = \frac{1}{5N}\sum_{j=1}^{n} ||x_i - x_j|| \tag{B.10.6}$$

wher d_{si} = sensing distance for the i-th krill, N = number of krills. Factor 5 is empirically obtained. If the distance of two krills is less than the defined sensing distance, they are neighbors. α_i^{target} is the effect of a krill with the best fitness on the i-th krill and leads to global optima. It is evaluated as follows:

$$\alpha_i^{\text{target}} = C^{\text{best}}\hat{K}_{i,\text{best}}\hat{x}_{i,\text{best}} \tag{B.10.7}$$

where $\hat{K}_{i,\text{best}}$ = best objective function value of the i-th krill, $\hat{x}_{i,\text{best}}$ = best position value of the i-th krill, C^{best} is the effective coefficient of the krill with the best fitness to the i-th krill and is evaluated as:

$$C^{\text{best}} = 2\left(rand + \frac{I}{I_{\max}}\right) \tag{B.10.8}$$

where $rand \in [0, 1]$ = random number, I = current iteration, $I_{\max}$ = maximum number of iterations.

(ii) **Foraging motion.** The foraging motion of the i-th krill is a factor of two parameters, first is the food location, and the second is the previous experience with the food location and is calculated as:

$$F_i = V_f \beta_i + w_f F_i^{\text{old}} \tag{B.10.9}$$

where where V_f = parameter for tuning the foraging speed, β_i = centroid location of the i-th krill, w_f = inertia weight of the foraging speed in the range $[0, 1]$, F_i^{old} = last foraging motion value for the i-th krill. The centroid location of the i-th krill is evaluated as follows:

$$\beta_i = \beta_i^{\text{food}} + \beta_i^{\text{best}} \tag{B.10.10}$$

where β_i^{best} = best objective function value for the i-th individual, β_i^{food} is centroid attractive of the i-th krill and is determined as follows:

$$\beta_i^{\text{food}} = C^{\text{food}} \hat{K}_{i,\text{food}} \hat{x}_{i,\text{food}} \tag{B.10.11}$$

$$C^{\text{food}} = 2\left(1 - \frac{I}{I_{\max}}\right) \tag{B.10.12}$$

where $\hat{K}_{i,\text{food}}$ and $\hat{x}_{i,\text{food}}$ is the normalized value of the objective function and the normalized value of the i-th centroid. The center of the individual's food for each iteration is calculated as:

$$x^{\text{food}} = \frac{\sum_{i=1}^{n} \frac{1}{K_i} x_i}{\sum_{i=1}^{n} \frac{1}{K_i}} \tag{B.10.13}$$

The effect of the best objective function value of the i-th krill is evaluated as:

$$\beta_i^{\text{best}} = \hat{K}_{i,\text{ibest}} \hat{x}_{i,\text{ibest}} \tag{B.10.14}$$

where $\hat{K}_{i,\text{ibest}}$ and $\hat{x}_{i,\text{ibest}}$ is the best previous objective function value, and the best previously visited centroid of the i-th krill. The movement induced by other krills and forging movement decrease with increasing iterations.

(iii) **Physical diffusion.** Physical diffusion is the net movement of each krill from high-density to low-density regions. Physical diffusion for the i-th krill is determined as:

$$D_i = D_{\max}\left(1 - \frac{I}{I_{\max}}\right)\rho \tag{B.10.15}$$

where $D_{\max}$ = parameter for tuning the diffusion speed, ρ refers to an array containing random values between $[1, 1]$.

(iv) **Updating the krills.** The motion is induced by other krills, foraging motion, and physical diffusion change each krill's position toward the best objective function using the following equation:

$$x_i(I+1) = x_i(I) + \Delta t \frac{dx_i}{dt} \tag{B.10.16}$$

$$\Delta t = C_t \sum_{i=1}^{n} (UB_i - LB_i) \tag{B.10.17}$$

where Δt = sensitive constant, n = total krills, LB_i and UB_i are the lower and upper bounds of the i-th individual, C_t = constant value between $[0, 2]$ used as a scale factor of the speed vector.

B.11. Multi-verse optimization [85]

B.11.1. Behavior

The multi-verse theory, which implies that multiple universes have their own physical laws and an inflation rate that causes their expansion in space.

B.11.2. Learning equations

Universes interact through white holes, black holes, and wormholes. White holes play a significant role in the birth of the universe and have a high inflation rate. Black holes have a colossal gravitational force that gulps everything inside, even light. Wormholes are tunnels through which objects can travel among universes or galaxies. The fitness function of each universe/solution is proportional to the inflation rate. Initially, universes are sorted according to their inflation rates and one is selected randomly through the roulette wheel mechanism to be a white hole. This is done by the following equation:

$$x_i^j = \begin{cases} x_k^j & r1 < NI(U_i) \\ x_i^j & r1 \geq NI(U_i) \end{cases} \tag{B.11.1}$$

where $U[n, d]$ = matrix represents the complete universe with all elements, d = total number of parameters, n = number of universes, x_i^j = j-th parameter in the i-th universe, NI is normalized inflation rate of the i-th universe, $r1 \in [0, 1]$ = random number, x_k^j = j-th parameter of the k-th universe chosen by roulette wheel. This allows universes to exchange objects and improve inflation rates. Things are also randomly exchanged between universes through wormholes, and it is assumed that wormholes are established between others and the best universe formed yet. This mechanism can be formulated as follows:

$$x_i^j = \begin{cases} \begin{cases} X_j + TDR \cdot ((ub_j - lb_j) \cdot r4 + lb_j) & r3 < 0.5 \\ X_j - TDR \cdot ((ub_j - lb_j) \cdot r4 + lb_j) & r3 \geq 0.5 \end{cases} & r2 < WEP \\ x_i^j & r2 \geq WEP \end{cases} \tag{B.11.2}$$

where X_j is the j-th parameter of the best universe obtained so far, ub_j and lb_j are the upper and lower bound values of the j-th variable, x_i^j is the j-th parameter of the i-th universe, and $r2, r3, r4$ are random numbers between $[0, 1]$, WEP and TDR are coefficients used in the equation and can be calculated using the formula:

$$WEP = \min + l \cdot \left(\frac{\max - \min}{L} \right) \tag{B.11.3}$$

$$TDR = 1 - \frac{l^{1/p}}{L^{1/p}} \tag{B.11.4}$$

where *WEP* (Wormhole Existence Probability) increases linearly over iterations to support exploitation, L = maximum number of iteration, l = current iteration, max and min are 1 and 0.2 by default, *TDR* (Traveling Distance Rate) defines the distance rate by which an object can be transferred to the best universe obtained yet, $p = 6$ is exploitation precision, the higher its value, the faster the exploitation.

B.12. Symbiotic organisms search [86]

B.12.1. Behavior

The symbiotic relationships that exist between paired organisms to survive in the ecosystem.

B.12.2. Learning equations

Each organism in an algorithm is a solution in the d-dimensional search space, and they are refined through three phases applied serially. The three phases are explained as follows:

(i) **Mutualism.** Mutualism relationship benefits both the interacting organisms. Let X_i is an organism in search space, and X_j is a random candidate interacting with i-th organism to increase mutual survival advantage. Their positions are updated according to the following equations:

$$X_i^* = X_i + rand(0,1) \cdot (X_{\text{best}} - \textit{Mutual Vector} \cdot BF_1) \tag{B.12.1}$$
$$X_j^* = X_j + rand(0,1) \cdot (X_{\text{best}} - \textit{Mutual Vector} \cdot BF_2) \tag{B.12.2}$$
$$\textit{Mutual Vector} = \frac{X_i + X_j}{2} \tag{B.12.3}$$

where *rand* is a random function that produces a number between 0 and 1, X_{best} is the best position searched by all organisms yet or best fitness value, BF_1 and BF_2 are the benefit factor of i-th and j-th organism, respectively. Their values are either 1 or 2.

(ii) **Commensalism.** Commensalism only benefits one organism, and the other one remains unaffected. The interaction between X_i and X_j is updated as follows:

$$X_i^* = X_i + rand(-1,1) \cdot (X_{\text{best}} - X_j) \tag{B.12.4}$$

where only X_i is benefited from the interaction while X_j neither benefits nor gets harmed from it. $(X_{\text{best}} - X_j)$ represents that benefit.

(iii) **Parasitism.** In this phase, only one candidate benefits and the relationship harms the other candidate. Parasite Vector is created in search space by copying X_i to interact with host X_j. Parasite Vector replaces X_j if it has better fitness value; otherwise, X_j survives.

$$X_j = \begin{cases} PV & \text{if } f(PV) > f(X_j) \\ X_j & \text{if } f(PV) \leq f(X_j) \end{cases} \tag{B.12.5}$$

where PV is a parasite vector and $f(\cdot)$ represents fitness function.

B.13. Flower pollination optimization [87]

B.13.1. Behavior

The reproduction mechanism of flowers through pollination in nature.

B.13.2. Exploration equations

When pollinators such as bees that can fly far and may portray Lévy flight behavior, contributes to cross-pollination and are considered global pollination. For single objective problems, it can be assumed that each plant has only one flower, and each flower has only one pollen, referred to as a solution x_i. Global pollination ensures the reproduction of the most fittest flowers and can be represented as g_*. Consistency of a flower is its reproduction probability and can be calculated using the following formula:

$$x_i^{t+1} = x_i^t + L(x_i^t - g_*) \tag{B.13.1}$$

where x_i^t is the solution vector at iteration t for pollen i and g_* is the best solution found yet. L is the step size drawn from Lévy flight distribution, it can be evaluated as:

$$L \sim \frac{\lambda\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi}\frac{1}{s^{1+\lambda}}, (s \gg s_0 > 0) \tag{B.13.2}$$

where $\Gamma(\lambda)$ is the standard gamma function with an index λ. This distribution works for large step sizes, $s > 0$.

B.13.3. Exploitation equations

Local pollination is when self-pollination happens through abiotic means. For local pollination, flower consistency is calculated as follows:

$$x_i^{t+1} = x_i^t + \epsilon(x_j^t - x_k^t) \tag{B.13.3}$$

where x_j^t and x_k^t are pollens from the different flowers of the same plant species. If x_j^t and x_k^t are selected from the same population and ϵ is from a uniform distribution in $[0, 1]$, this equation will represent the local random walk. The switch between local and global pollination is controlled by parameter p which ranges between $[0, 1]$.

B.14. Teaching learning based optimization [88]

B.14.1. Behavior

The traditional teaching-learning phenomenon of a classroom.

B.14.2. Learning equations

The teacher tries to train learners in the best way possible to increase their level of knowledge. All the learners are the population and the teacher is the best solution. Design variables are different subjects taught in the class, and the result is analogous to the fitness value. Teaching is done in two phases:

(i) **Teacher Phase.** Learners learn from best learner, i.e., teacher. Teacher tries to elevate the mean of class, best to his abilities. Let, M_i be the mean of class result and T_i be the teacher at any iteration i. T_i will try to improve M_i to M_{new}. So, new solution is updated according to the following equation:

$$X_{\text{new},i} = X_{\text{old},i} + r_i(M_{\text{new}} - T_F M_i) \tag{B.14.1}$$

where $r_i \in [0,1]$ = random number, T_F is a teaching factor that can have value either 1 or 2. It is a heuristic step and decided randomly with equal probability as $T_F = \text{round}(1 + rand(0,1)\{2-1\})$.

(ii) **Learner Phase.** Learners learn from teachers or among themselves through presentations, discussions or formal communication. A learner will gain knowledge if another learner has more knowledge than him. For leaner X_i in the class, the updating mechanism is as follows:

$$newX_i = \begin{cases} X_i + rand \cdot (X_i - X_k) & f(X_i) < f(X_k) \\ X_i + rand \cdot (X_k - X_i) & \text{otherwise} \end{cases} \tag{B.14.2}$$

where $newX_i$ = new positions of the i-th learner X_i, X_k = random learner from the class, $f(X_i)$ = fitness values of the learner X_i, $f(X_k)$ = fitness values of the learner X_k, $rand \in [0,1]$ = random number.

B.15. Gravitational search optimization [89]

B.15.1. Behavior

Newton's law of gravity and the second law of motion.

B.15.2. Learning equations

Each agent has its Position X, Active Gravitational Mass (M_a), Passive Gravitational Mass (M_p), and Inertial Mass (M_i). The position is the solution of the problem and masses are evaluated using the fitness function. Gravitational force applies to all agents; thus global movement of all agents is forced towards heavier masses supporting exploitation and an optimum solution in the search space. The system of N agents with their initial positions is defined as follows:

$$X_i = (x_i^1, x_i^2, \ldots, x_i^d, \ldots, x_i^n) \text{ for } i = 1, 2, \ldots, N. \tag{B.15.1}$$

where x_i^d is the position of the i-th search agent in the d-th dimension. At any given time t, the gravitational force acting between agent i and agent j is:

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t) \cdot M_{aj}(t)}{R_{ij}(t) + \epsilon}(x_j^d(t) - x_i^d(t)) \tag{B.15.2}$$

where M_{pi} = passive gravitational mass of the i-th object, M_{aj} = active gravitational mass of the j-th agent, ϵ is a constant, $G(t)$ = gravitational constant at time t, $R_{ij}(t)$ is the Euclidean distance between two agents i and j, can be calculated as:

$$R_{ij}(t) = \sqrt{\sum_{k=1}^{n}(X_{ik}(t) - X_{jk}(t))^2} \tag{B.15.3}$$

To support exploration, a random factor is added to the total force acting on an agent. It can be represented as:

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j \; F_{ij}^d(t) \tag{B.15.4}$$

where $rand_j$ is a random number between $[0, 1]$. According to the law of motion, acceleration of the i-th agent in the d-th dimension at time t is:

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)} \tag{B.15.5}$$

where M_{ii} is the inertial mass of the i-th agent. The next velocity and position of i-th agent can be updated using following formulas:

$$v_i^d(t+1) = rand_i \cdot v_i^d(t) + a_i^d(t) \tag{B.15.6}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{B.15.7}$$

where $rand_i$ is a uniform random number in between $[0, 1]$. The gravitational constant will be reduced over time to control the search accuracy. If it is assumed that gravitational and inertial masses are equal, they can be updated as follows using the map of fitness:

$$M_{ai} = M_{pi} = M_{ii} = M_i, i = 1, 2, \ldots, N. \tag{B.15.8}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \tag{B.15.9}$$

$$m_i(t) = \frac{\text{fit}_i(t) - \text{worst}(t)}{\text{best}(t) - \text{worst}(t)} \tag{B.15.10}$$

$$\text{best}(t) = \min_{j\in\{1,2,\ldots,N\}} \text{fit}_j(t) \tag{B.15.11}$$

$$\text{worst}(t) = \max_{j\in\{1,2,\ldots,N\}} \text{fit}_j(t) \tag{B.15.12}$$

where $\text{fit}_i(t)$ is the fitness value of the i-th agent at time t. The number of agents over iteration reduces to maintain a robust balance between exploration and exploitation, so only heavy mass agents apply their force on other agents stored as $Kbest$. It is the function of time, and has an initial value of K_0 which decreases with time. Initially, all search agents

apply the force, and with each iteration, $Kbest$ is linearly reduced and in the end, there will be just one agent applying force to the others. Therefore, force is updated as:

$$F_i^d(t) = \sum_{j \in Kbest, j \neq i} rand_j \; F_{ij}^d(t) \tag{B.15.13}$$

where $Kbest$ is the set of first K agents with the best fitness value and biggest mass.

B.16. Biogeography-based optimization [90]

B.16.1. Behavior

The migration of species between islands.

B.16.2. Learning equations

Habitats with good and favorable living conditions have high Habitat Suitability Index (HSI), represents good solutions and have high emigration rate and low immigration rate. Islands with low HSI represent poor solutions but have a high immigration rate due to their sparse species count and low emigration rate. Immigration and emigration rates are fitness functions of a habitat. The factors influencing HSI are called Suitability Index Variables (SIVs) and are considered to be the independent variables of the habitat. The algorithm has two main steps, migration and mutation.

(i) **Migration.** It is a probabilistic operator that improves a habitat H_i. Each habitat's migration rate is used to share features between habitats. For each habitat H_i, its immigration rate (λ_i) is used to decide whether or not to immigrate. If immigration is selected, the emigrating habitat H_j is selected probabilistically based on the emigration rate (μ_i). Rates and Migration are defined as follows:

$$\mu_i = \frac{Ei}{N} \tag{B.16.1}$$

$$\lambda_i = I\left(1 - \frac{i}{N}\right) \tag{B.16.2}$$

$$H_i(SIV) \leftarrow H_j(SIV) \tag{B.16.3}$$

where N is the total population size.

(ii) **Mutation.** It is a probabilistic operator that randomly modifies a habitat's SIV based on the habitat's a priori species count probability. The purpose of mutation tends to increase diversity among the population. For low HSI solutions, mutation gives them a chance to enhance the quality of solutions, and for high HSI solutions, the mutation can improve them even more than they already have.

e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240108, DOI: 10.37190/e-Inf240108

Automated Code Reviewer Recommendation for Pull Requests

Mina-Sadat Moosareza*, Abbas Heydarnoori**
*Independent Researcher
**Department of Computer Science, Bowling Green State University
m.s.moosareza@gmail.com, aheydar@bgsu.edu

Abstract

minu.15em **Background:** With the advent of distributed software development based on pull requests, it is possible to review code changes by a third party before integrating them into the master program in an informal and tool-based process called Modern Code Review (MCR). Effectively performing MCR can facilitate the software evolution phase by reducing post-release defects. MCR allows developers to invite appropriate reviewers to inspect their code once a pull request has been submitted. In many projects, selecting the right reviewer is time-consuming and challenging due to the high requests volume and potential reviewers. Various recommender systems have been proposed in the past that use heuristics, machine learning, or social networks to automatically suggest reviewers. Many previous approaches focus on a narrow set of features of candidate reviewers, including their reviewing expertise, and some have been evaluated on small datasets that do not provide generalizability. Additionally, it is common for them not to meet the desired accuracy, precision, or recall standards.

Aim: Our aim is to increase the accuracy of code reviewer recommendations by calculating scores relatively and considering the importance of the recency of activities in an optimal way.

Method: Our work presents a heuristic approach that takes into account both candidate reviewers' expertise in reviewing and committing, as well as their social relations to automatically recommend code reviewers. During the development of the approach, we will examine how each of the reviewers' features contributes to their suitability to review the new request.

Results: We evaluated our algorithm on five open-source projects from GitHub. Results indicate that, based on top-1 accuracy, top-3 accuracy, and mean reciprocal rank, our proposed approach achieves 46%, 75%, and 62% values respectively, outperforming previous related works.

Conclusion: These results indicate that combining different features of reviewers, including their expertise level and previous collaboration history, can lead to better code reviewer recommendations, as demonstrated by the achieved improvements over previous related works.

Keywords: Automated code reviewer recommendation, Modern code review, Heuristic algorithms.

Submitted: 20 Dec. 2023; Revised: 24 Mar. 2024; Accepted: 5 May 2024; Available online: 31 Jul. 2024

1. Introduction

The process of code review previously involved third parties inspecting code in face-to-face meetings before integrating code into the master branch to find and fix any problems. Due to its potential to significantly impact post-release quality, it has drawn considerable attention in the software industry. It is becoming more common to observe software systems, especially open-source ones, being developed by programmers from different geographic locations. There are powerful tools, such as GitHub, supporting this process by the means of pull requests. A pull request is an event where a contributor develops a code change and requests owners to merge it with the main program. This new program development method requires a different style of code review, called Modern Code Review (MCR). The process of MCR is tool-based and informal, where the developer invites the appropriate reviewer to inspect the code for integration after sending a pull request.

The code review process is costly because the reviewer must read, understand, and critique the code. For this reason, the author is recommended to select programmers knowledgeable about and capable of analyzing the modified sections. This is difficult and time-consuming as the pull requests volume of many projects is high. Furthermore, determining who is the most appropriate reviewer for a new pull request is more challenging in MCR since project participants' capabilities are unknown. Study [1] found that 4% to 30% of code reviews have problems assigning reviewers, in which case it takes 12 additional days for changes to be approved. Meanwhile, [2] suggests that reviewing new code changes can reduce post-release defects of software and thus improve its quality. Therefore, assigning code reviewers is a major problem in software engineering, and automated reviewer recommendations could be very useful.

There have been a variety of approaches (e.g., [2–4]) proposed to automatically suggest appropriate reviewers in order to resolve the above issue. The following factors are primarily considered for this purpose: reviewer *expertise* in terms of her previous reviews or commits on changed files (e.g., [3, 5]), number of reviewer collaborations with the new pull request author in the project (e.g., [6, 7]), reviewer *activity* based on the number of previous reviews and commits the reviewer has made on all project files (e.g., [8]), and *workload* based on the number of open reviews assigned to her (e.g., [9]). The majority of previous studies have focused on just one or two of the features mentioned above. Our experimental evaluations did indicate, however, that combining these features would allow us to recommend more appropriate reviewers.

This observation led us to develop a heuristic-based approach to recommending reviewers. Our algorithm calculates three different scores for each candidate reviewer: *review expertise*, *commit expertise*, and *collaboration history*. Then, we combine them using a weighted sum. Our experiments determine the relative importance of each feature in selecting an appropriate reviewer based on these weights. Previous approaches also sometimes are evaluated on small datasets, which negatively impacts the generalizability of the results. To mitigate this issue, we evaluate our proposed algorithm on a large dataset of five GitHub projects [10]. We show that our proposed approach achieves 46%, 85%, and 92% in terms of top-1 accuracy, top-3 accuracy, and mean reciprocal rank, respectively. In summary, this paper has the following contributions:

– Introducing a new heuristics-based approach to automate reviewer recommendations for pull requests.
– Taking into account all three feature categories of review expertise, commit expertise, and collaboration.

– Quantitatively evaluating the proposed approach and indicating that it outperforms existing approaches.

This paper is organized as follows. In Section 2, we explain our new algorithm for code reviewer recommendation. Section 3 shows the evaluation setup, qualitative and quantitative results, and threats to validity. In Section 4, we discuss the pros and cons of our proposed approach. Section 5 reviews related work, and a conclusion and upcoming direction are presented in Section 6.

2. Proposed approach

Figure 1 illustrates the overall process of our heuristic solution. The previous review recommenders considered a variety of features when selecting candidates. Our approach takes expertise and collaboration into account, and we will report that the activity feature

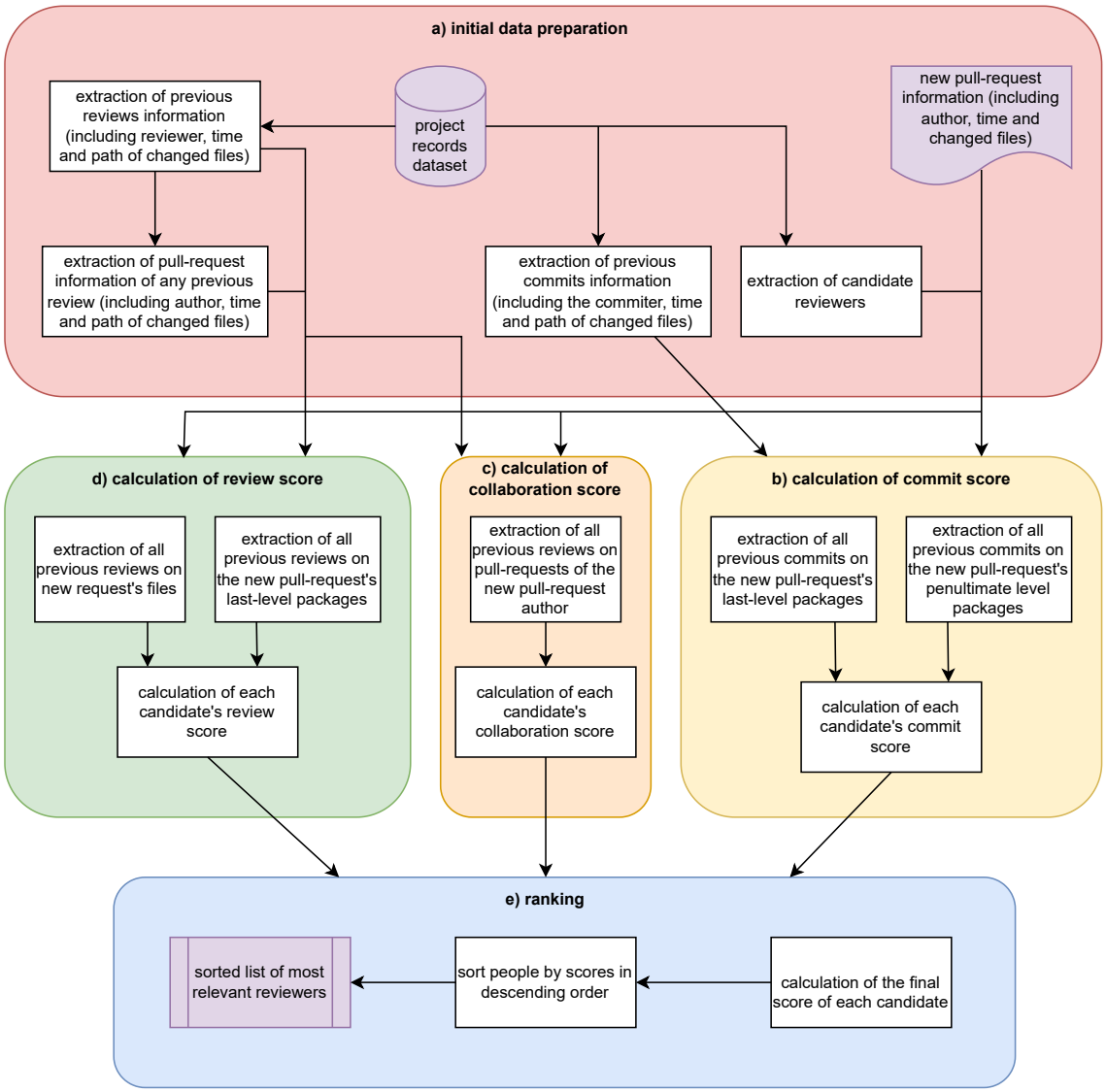


Figure 1. Overview of the proposed approach

was excluded from our algorithm as it had not provided any significant improvements. Initially, the primitive data needed for subsequent steps are extracted from the existing dataset as shown in section (a). Following this, each person's score is calculated in three parallel processes (b), (c), and (d) considering their experience of commitment, review, and collaboration. The final score of each candidate is determined by combining all these scores in section (e). Afterward, the output will be a list of candidates sorted by their final scores.

We had to examine the effect of each change on the original formula during the development of our heuristic algorithm. These intermediate evaluations were conducted using the dataset provided by [10], which includes five GitHub projects. Details of this dataset are provided in Section 3.

2.1. Scores definition

This study focuses on three categories of features; collaboration, activity, and expertise, including experience of review and commitment, which will be briefly described below:

- **Review score.** Most reviewer recommendation methods take into account the reviewer's experience, and its importance has always been emphasized. In [11], the authors provide a review ownership criterion. They demonstrated that if developers participate actively in reviews, they could specialize in related code snippets. Several reviewer recommendation systems, such as those presented in [3, 5, 12], are based on review experience, and its importance has been highlighted in articles [13–15].
- **Commit score.** The authors of [16] indicated that developers who make plenty of changes to portions of a program code should be considered owners of those parts. The findings of [17] suggest that individuals are more likely to review changes they have experience with. As part of their proposed approaches, some previous automated reviewer recommendation systems, such as [5, 8, 9], have also considered commit experience. Also, in [15] it is mentioned that the level of the reviewer's commit experience impacts the usefulness of the review.
- **Collaboration score.** Collaboration score refers to a candidate reviewer's involvement with previous pull requests made by the new request's author. In some articles in the reviewer's recommendation area, including [17, 18], the importance of relationships between the candidate reviewer and the pull request author is emphasized in addition to expertise. Some of the previous automated reviewer recommendation approaches, such as the articles [6, 7, 9] approaches, have also benefited from collaboration score in addition to expertise. As a result, some previously automated reviewer recommendation approaches, such as [6, 7, 9], included collaboration scores along with expertise scores.
- **Activity score.** Lastly, some articles, such as [8] which proposed a way to recommend code reviewers, introduced a score that can be called activity. Level of activity indicates how actively a candidate participated in review processes throughout the project.

2.2. Initial formulation of scores

The first step was to calculate each score based on a relation introduced in previous articles. We then made appropriate changes to these relationships to improve them gradually during the steps described in the following subsections. In this research, the relationships presented

in [5] are used to calculate the initial score of review and commit based on (1) and (2), respectively:

$$\text{ReviewScore}(r) = \sum_{f \in F} n_{\text{review}}(r, f) \frac{1}{t_{\text{review}}(r, f)} + \sum_{d \in D} n_{\text{review}}(r, d) \frac{1}{t_{\text{review}}(r, d)} \quad (1)$$

$$\text{CommitScore}(r) = \sum_{f \in F} n_{\text{change}}(r, f) \frac{1}{t_{\text{change}}(r, f)} + \sum_{d \in D} n_{\text{change}}(r, d) \frac{1}{t_{\text{change}}(r, d)} \quad (2)$$

assuming F is the set of files changed in the new pull request, while D represents the set of last-level parent directories. $n_{\text{review}}(r, f)$ and $n_{\text{change}}(r, f)$ indicate how many reviews and commits were done by reviewer r for file f, whereas $n_{\text{review}}(r, d)$ and $n_{\text{change}}(r, d)$ refer to the number of previous reviews and commits that r performed on the files in directory d, respectively. Furthermore, $t_{\text{review}}(r, f)$ and $t_{\text{change}}(r, f)$ reflect the time elapsed since reviewer r last reviewed and committed file f, while $t_{\text{review}}(r, d)$ and $t_{\text{change}}(r, d)$ represent how long has passed since the last review and commit on directory d by reviewer r, respectively. In the fractions of the final score, these time coefficients appear at the denominators, so that as time passes since a review or commit, its effect decreases.

To calculate the initial score of collaboration, we used the relation presented in [6], which is calculated according to the (3):

$$\text{CollaborationScore}(r) = n_{\text{collaboration}}(r, a) \quad (3)$$

where a refers to the author of the new pull request, and $n_{\text{collaboration}}(r, a)$ is the number of collaborations between a and reviewer r, i.e., the number of pull requests authored by a, and reviewed by r.

To calculate the initial activity score, we used the relation presented in [8], which is calculated according to the (4):

$$\text{ActivityScore}(r) = \sum_{t < 360} n_{\text{review}}(r, t) \quad (4)$$

where $\sum_{t<360} n_{\text{review}}(r, t)$ means the number of times which r has reviewed a pull request of the whole project in the past year.

2.3. Relative formulation of reviews and commit scores

Assume there are two cases in which reviewer r reviewed file f twice and last reviewed it one day ago, but in one case, the file has already been reviewed twice in total, while in the other, it has been reviewed a hundred times. In both cases the value of $\frac{n_{\text{review}}(r, f)}{t_{\text{review}}(r, f)}$ will be equal to 2. However, we know that the level of expertise of the reviewer r on the file f relative to all project reviewers is much higher in the first case than in the second one. Because in the first case, 100% of the review history of f belongs to r, but in the second, this value is only two percent, and there may be people with more experience on f in the project. Therefore, like [3, 8], we decided to calculate the review and commit experience

score in relative terms as (5) and (6), respectively:

$$\begin{aligned}\text{ReviewScore}(r) = &\sum_{f \in F} \frac{n_{\text{review}}(r,f)}{\sigma_{r' \in R} n_{\text{review}}(r',f)} \cdot \frac{1}{t_{\text{review}}(r,f)} + \\ &+ \sum_{d \in D} \frac{n_{\text{review}}(r,d)}{\sigma_{r' \in R} n_{\text{review}}(r',d)} \cdot \frac{1}{t_{\text{review}}(r,d)}\end{aligned} \tag{5}$$

$$\begin{aligned}\mathit{CommitScore}(r) = &\sum_{f \in F} \frac{n_{\text{change}}(r,f)}{\sigma_{r' \in R} n_{\text{change}}(r',f)} \cdot \frac{1}{t_{\text{change}}(r,f)} + \\ &+ \sum_{d \in D} \frac{n_{\text{change}}(r,d)}{\sigma_{r' \in R} n_{\text{change}}(r',d)} \cdot \frac{1}{t_{\text{change}}(r,d)}\end{aligned} \tag{6}$$

where R is the list of all people who have provided reviews for the project, and $\sigma_{r' \in R} n_{\text{review}}(r', f)$ and $\sigma_{r' \in R} n_{\text{review}}(r', d)$, respectively, denote all reviews on file f and directory d. Also, $\sigma_{r' \in R} n_{\text{change}}(r', f)$ and $\sigma_{r' \in R} n_{\text{change}}(r', d)$ indicate how many commits are done on file f and directory d, respectively. We examined both absolute and relative formulas separately on our dataset to test the validity of our hypothesis regarding the importance of the relative review experience of candidates. The observations showed an improvement in the results of the review and commit scores in the relative state. Relative calculation of collaboration and activity scores is meaningless because they are calculated regardless of the files under review.

2.4. Applying the impact of time factor

We have already seen that the time factor is at the denominator of the fractions in the initial review score formula, to reduce the impact of older reviews. However, the relationship between the time elapsed since candidates' last review and their expertise is not always linear. After reviewing a file, at first, a person's level of expertise in the file decreases dramatically with each passing day, as a result of him slowly forgetting the contents of the file, and on the other hand, the content of the file altered by others over time. However, if a person has not reviewed a file for a long time, such as a year, the daily passage of time does not have a significant impact on his or her level of expertise because he or she has probably forgotten most of its contents and the file has changed more frequently.

Therefore, instead of using a linear relationship to calculate the effect of time in the denominator of fractions, we can use a relationship whose slope is initially high and then gradually decreases. This can be accomplished by considering relations like logarithms or second or higher roots for the time at the denominator. This explanation holds for the scores for collaboration, commitment, and activity, as well. We used our dataset to test the scoring formula by substituting different time coefficients since it has only been considered linearly in previous articles. For all scores, we found that the presence of the third root of the time in the denominator provided the best results. There is an average accuracy deduction with higher roots. Therefore, the formulas for calculating review, commit, collaboration, and activity scores are modified as (7), (8), (9) and (10), respectively:

$$\text{ReviewScore}(r) = \sum_{f \in F} \frac{n_{\text{review}}(r,f)}{\sigma_{r' \in R} n_{\text{review}}(r',f)} \cdot \frac{1}{\sqrt[3]{t_{\text{review}}(r,f)}} +$$

$$+ \sum_{d \in D} \frac{n_{\text{review}}(r,d)}{\sigma_{r' \in R} n_{\text{review}}(r',d)} \cdot \frac{1}{\sqrt[3]{t_{\text{review}}(r,d)}} \tag{7}$$

$$\text{CommitScore}(r) = \sum_{f \in F} \frac{n_{\text{change}}(r,f)}{\sigma_{r' \in R} n_{\text{change}}(r',f)} \cdot \frac{1}{\sqrt[3]{t_{\text{change}}(r,f)}} +$$

$$+ \sum_{d \in D} \frac{n_{\text{change}}(r,d)}{\sigma_{r' \in R} n_{\text{change}}(r',d)} \cdot \frac{1}{\sqrt[3]{t_{\text{change}}(r,d)}} \tag{8}$$

$$\text{CollaborationScore}(r) = n_{\text{collaboration}}(r,a) \cdot \frac{1}{\sqrt[3]{t_{\text{collaboration}}(r,a)}} \tag{9}$$

$$\text{ActivityScore}(r) = (\sum_{t<360} n_{\text{review}}(r,t) \cdot \frac{1}{\sqrt[3]{t_{\text{review}}(r)}} \tag{10}$$

In which $t_{\text{collaboration}}(r,a)$ and $t_{\text{review}}(r)$ refer to the time elapsed since the last review of r on the pull requests of the author of the new request and the project as a whole, respectively.

2.5. Examining reviews and commit scores for directories of previous levels

Using the track record of candidates committing and reviewing on the last-level directories, [5] encouraged us to test the effect on earlier-level directories as well. Considering review scores for directories at earlier levels did not improve the results, according to our experimental findings. On the other hand, the penultimate and last-level directories yielded the best results for the commit score.

The following may be the reason for the better results of commit experience on penultimate level directories over files. In a development environment, it is common for developers to repeatedly make changes to the same file once they have committed to it and familiarized themselves with its content. So, in many cases, the person who has the most experience committing on the file is the modifier of the pull request and should not be chosen as a reviewer. Additionally, files in the same path as a file usually contain similar content. The committers of these files may therefore have become acquainted with the file content during their committing. So, we changed the calculation formula for the commit score to be as (11):

$$\text{CommitScore}(r) = \sum_{d_1 \in D_1} \frac{n_{\text{change}}(r,d_1)}{\sigma_{r' \in R} n_{\text{change}}(r',d_1)} \cdot \frac{1}{\sqrt[3]{t_{\text{change}}(r,d_1)}} +$$

$$+ \sum_{d_2 \in D_2} \frac{n_{\text{change}}(r,d_2)}{\sigma_{r' \in R} n_{\text{change}}(r',d_2)} \cdot \frac{1}{\sqrt[3]{t_{\text{change}}(r,d_2)}} \tag{11}$$

where D_1 refers to the last-level directories of files that changed in the new pull request. As well, D_2 means the set of changed files' parent directories in the level preceding the last level.

2.6. Determining Optimal Coefficients

The work presented in [5] and other articles which combine different scores by adding values, such as [3, 9], assume the same coefficients for all sentences. However, this assumption is not necessarily correct. Thus, we performed multiple experiments to obtain relative coefficients for each of the scores while combining them to optimize the results. We derived the coefficients through iterative experimentation, selecting those that yielded the most favorable experimental outcomes. AT first, we assigned a coefficient of one to the commit score, as among the various scores calculated, it exhibited the weakest correlation with reviewer suitability for new request reviews. Subsequently, we explored different coefficients for the review score, evaluating the list of recommended reviewers and their Mean Reciprocal Rank (MRR) in each instance. Accordingly, we calculated the appropriate coefficients for the two-sentence scores of review and commit, resulting in (12) and (13), respectively.

$$\begin{aligned}\text{ReviewScore}(r) = 1.25 \cdot \sum_{f \in F} \frac{n_{\text{review}}(r,f)}{\sigma_{r' \in R} n_{\text{review}}(r',f)} \cdot \frac{1}{\sqrt[3]{t_{\text{review}}(r,f)}} + \\ + \sum_{d \in D} \frac{n_{\text{review}}(r,d)}{\sigma_{r' \in R} n_{\text{review}}(r',d)} \cdot \frac{1}{\sqrt[3]{t_{review}(r,d)}}\end{aligned} \tag{12}$$

$$\begin{aligned}\text{CommitScore}(r) = \sum_{d_1 \in D_1} \frac{n_{\text{change}}(r,d_1)}{\sigma_{r' \in R} n_{\text{change}}(r',d_1)} \cdot \frac{1}{\sqrt[3]{t_{\text{change}}(r,d_1)}} + \\ + 1.25 \cdot \sum_{d_2 \in D_2} \frac{n_{\text{change}}(r,d_2)}{\sigma_{r' \in R} n_{\text{change}}(r',d_2)} \cdot \frac{1}{\sqrt[3]{t_{\text{change}}(r,d_2)}}\end{aligned} \tag{13}$$

Next, we employed a similar approach to determine the coefficient for the collaboration score. We combined the review and commit scores, then combined them with the scores of collaboration and activity, determining the optimal coefficient for each. As for the activity score, we found that the best result was achieved with a coefficient of about 0.002. Therefore, the impact of the activity score is very small compared with the combination of the commit, review, and collaboration scores. Additionally, adding the activity score improved the results in only three out of five projects. With these observations, we can discard activity score when formulating heuristic relation and rely on the scores of expertise, including review and commitment, as well as collaboration. This is especially true when we observe that the average improvement of the result is only about 0.001, which is very small. Finally, the following relation (14) is our proposed heuristic relation:

$$\begin{aligned}\text{ReviewerScore}(r) = 7.7344 \cdot \sum_{f \in F} \frac{n_{\text{review}}(r,f)}{\sigma_{r' \in R} n_{\text{review}}(r',f)} \cdot \frac{1}{\sqrt[3]{t_{\text{review}}(r,f)}} \\ + 6.1875 \cdot \sum_{d_1 \in D_1} \frac{n_{\text{review}}(r,d_1)}{\sigma_{r' \in R} n_{\text{review}}(r',d_1)} \cdot \frac{1}{\sqrt[3]{t_{\text{review}}(r,d_1)}} \\ + 2.25 \cdot \sum\nolimits_{d_1 \in D_1} \frac{n_{\text{change}}(r,d_1)}{\sigma_{r' \in R} n_{\text{change}}(r',d_1)} \cdot \frac{1}{\sqrt[3]{t_{\text{change}}(r,d_1)}}\end{aligned}$$

$$+ 2.8125 \cdot \sum_{d_2 \in D_2} \frac{n_{\text{change}}(r, d_2)}{\sigma_{r' \in R} n_{\text{change}}(r', d_2)} \cdot \frac{1}{\sqrt[3]{t_{\text{change}}(r, d_2)}}$$

$$+ n_{\text{collaboration}}(r, a) \cdot \frac{1}{\sqrt[3]{t_{\text{collaboration}}(r, a)}} \tag{14}$$

3. Evaluations

In this section, we present our approach evaluation setup and report the results. We then compare it to previous related works and find that the proposed approach performs better.

3.1. Evaluations questions

We evaluated our approach to answer the following research questions:
1. What is the accuracy, precision, and recall of our code reviewer's recommendations?
2. Regarding the MRR of our approach, how is the quality of code reviewers ranking?
3. Is our algorithm superior to previous related works in terms of functionality and results?

3.2. Evaluations setup

The first step toward evaluating our approach is to submit a new pull request to our system as input. Having calculated the scores, the system produces the ranked list of reviewers as the output. Afterward, we evaluate the results using a set of criteria derived from the literature. The more actual reviewers there are on our suggested list, the better the algorithm functions.

We have written a code that completely automates our evaluations. Therefore, the personal opinions of evaluators do not affect the evaluation results. Furthermore, we compare our heuristic algorithm's results with those of previous related work to better judge its effectiveness. We will present the dataset, the evaluation criteria, and the previous work we chose to compare with our method results in the following subsections.

3.2.1. Dataset

Our analysis and evaluation of the proposed solution is based on the publicly available data presented in [10]. Due to its comprehensiveness and the fact that it covers a long period, the dataset is reliable and usable. The entire dataset is derived from nine projects on GitHub and Gerrit. The dataset we used for our research is only from its GitHub projects. There are records of five projects starting with their first pull request up until early 2020 in our dataset. As a result of the different sizes and languages used in these projects, the dataset has a favorable generality. There are only about 1 200 pull requests in the Zookeeper project, for instance. Meanwhile, there are more than 27 000 pull requests in the Spark project. In addition, on average, selected projects on GitHub have around sixteen thousand stars and more than ten thousand forks, which indicates high credibility and popularity.

This dataset contains no review date. Using the GitHub Rest API, we compiled the review dates for all five projects' pull requests. Our final dataset includes the author name,

Table 1. Specification of dataset used for evaluation

Project name	Number of total pull requests	Number of test pull requests	Number of candidate reviewers
Zookeeper	780	156	194
Kafka	5 492	549	877
Beam	6 966	697	761
Flink	5 928	593	925
Spark	16 977	1 698	2 369

Table 2. Minimum, maximum, mean and median number of reviewers for pull requests

Project name	Minimum number of reviewers	Maximum number of reviewers	Mean number of reviewers	Median number of reviewers
Zookeeper	1	6	1.98	2
Kafka	1	17	1.84	2
Beam	1	11	1.37	1
Flink	1	9	1.44	1
Spark	1	14	1.98	2

creation date, and path of modified files in each pull request, the author name, the modified files, and the commit date for each commit, as well as the reviewer name, review date, and pull request number for each review. All individuals who reviewed or committed to the project before the new pull request was made, were considered candidates for review.

First, we set aside pull requests that have not been reviewed; because a recommended reviewer must have actually reviewed the pull request for the recommendation to be correct. Therefore, we cannot use these requests in the evaluation process since their correct answer is unknown. As in previous articles, such as [1, 3], we selected some late pull requests for each project and implemented our proposed solution to anticipate who would be the best reviewers for these requests. Data from previous pull requests were also used to calculate candidates' records. In Table 1, we present the number of pull requests selected, and the number of candidate reviewers. For more information, in Table 2, we provide the values for the minimum, maximum, mean and median number of reviewers for pull requests used in experiments.

3.2.2. Evaluation criteria

As defined in previous works, such as [1, 5, 6, 10] the proposed reviewer of a recommendation system is valid when that reviewer has actually reviewed the new pull request. We leveraged the criteria used in the previous automated reviewer recommendation studies as correct and valid criteria that measure the accuracy of the approaches. Many previous works, such as [3, 5–7], have used precision@m, recall@m, and f_score@m to evaluate their solutions. In precision@m, we can see what proportion of the candidates we proposed in the final list of reviewers have actually reviewed the pull request. Recall@m shows what proportion of real reviewers is on our suggested list. The f_score@m also combines precision and recall and assigns a final score to the list of recommended reviewers for each pull request. The methods of calculating these criteria are given in the (15), (16), and (17), respectively.

$$\text{precision@m} = \frac{|RR(p) \cap AR(p)|}{|RR(p)|} \tag{15}$$

$$\text{recall@m} = \frac{|RR(p) \cap AR(p)|}{|AR(p)|} \tag{16}$$

$$\text{f\_score@m} = 2 \cdot \frac{\text{precision@m} \cdot \text{recall@m}}{\text{precision@m} + \text{recall@m}} \tag{17}$$

where p, $RR(p)$, and $AR(p)$ refer to the new pull request, the recommended reviewers set for it and its actual reviewers set, respectively. m also indicates the number of reviewers at the beginning of the list for whom we computed the criterion. As an example, if $m = 3$, the values of these criteria will be calculated for the first three individuals on the list. To match all previous work, such as [3, 6, 7], we set the value of m in our evaluations equal to 1, 2, 3, 5, and 10. Articles of the other group, such as [1, 10, 19], have used the top-k accuracy criterion to measure their solutions. Top-k accuracy indicates for what proportion of the pull requests the first k reviewers in the proposed list included at least one actual reviewer. This criterion is calculated according to (18):

$$\text{top-}k \text{ accuracy}(P) = \frac{\sum_{p \in P} \text{isCorrect}(p, \text{top-}k)}{|P|} \tag{18}$$

where P is the set of pull requests and k is the number of proposed reviewers we have considered in the calculation. Following previous works, such as [1, 10, 19], we considered the values of k in our evaluations to be 1, 3, 5, and 10, respectively. The Mean Reciprocal Rank (MRR) is also widely used in this area. The mean reviewer rank is calculated as the average of the inverse of the first rank of the reviewers who actually reviewed the new pull request. Based on this criterion, we determine how much we have to scroll down the list of recommended reviewers to locate the first reviewer who is suitable. It is calculated as (19):

$$MRR = \frac{1}{|n|} \sum_{i=1}^{|n|} \frac{1}{\text{rank}_i} \tag{19}$$

where n is the total number of pull requests and $rank_i$ is the rank of the first correct answer in the proposed reviewers list for the i-th pull request. We have used these criteria to evaluate our approach.

3.2.3. Comparison with Related Works

Previous reviewer recommenders are not easy to judge and it is not easy to conclude that one method always achieves the highest accuracy. According to the experimental studies of [20], each of these methods is most effective for a particular dataset. To evaluate our approach, we selected the WhoDo reviewer recommender to compare our approach results with. WhoDo is a more recent approach with better results compared to more popular rivals such as Revfinder and CHRev, and the most similar heuristic method to ours. It is published in the proceedings of a well-known, high-rank conference. WhoDo has two versions: The first version only considers the expertise score of the candidates based on their commitment and review records, whereas the second version also takes into account the workload score to balance the different reviewers' workloads. It is pointed out in this work that the system may recommend reviewers who have not reviewed the pull request if the workload is taken into account. Developers tend to choose reviewers based primarily

on their level of expertise since they do not know the workload of candidates. Therefore, naturally, the precision and recall of the second version of WhoDo are less than those of its first version. Hence, to demonstrate its functionality fairly, we need to compare our algorithm with WhoDo's first version since we did not consider workload. To do so, we reimplemented WhoDo first version heuristic approach according to authors explanations and using provided formulas.

The WhoDo developers showed that the second version is comparable to CHRev's introduced in [3] in terms of precision and recall. Furthermore, the authors of [3] found their system to be superior to previous methods in evaluations. Hence, by demonstrating the superiority of our approach over WhoDo's first version, we are showing that our approach is better than most previous efforts. Moreover, WhoDo was introduced in 2019 and is thus newer than many other approaches. These were the reasons why we chose WhoDo to compare our approach with.

3.3. Evaluations results

3.3.1. Quantitative results

Following the reimplementing of WhoDo, we calculated the precision@m, recall@m, and f_score@m introduced in the previous section for our algorithm and the first version of WhoDo, the results of which can be found in Table 3. Due to the fact that we utilize the Review, Commit, and Collaboration scores to find the appropriate reviewer, RCC-Finder is the name we gave our approach. RCC-Finder's precision and recall are shown by Table 3, answering the research question RQ1.

Here it is appropriate to see the raw values of Table 3 in a visual graph to see the resulting improvements more clearly. This was done by calculating precision@m, recall@m, and f_score@m for WhoDo and RCC-Finder across all five projects for integers ranging

Table 3. Comparison of Precision@m, Recall@m and F_score@m for Whodo and RCC-Finder

Project name	Zookeeper		Kafka		Beam		Flink		Spark		Average	
Approach	Whodo	RCC-Finder	Whodo	RCC-Finder	Whodo	RCC-Finder	Whodo	RCC-Finder	Whodo	RCC-Finder	Whodo	RCC-Finder
Precision@1	0.44	0.56	0.35	0.41	0.27	0.39	0.41	0.56	0.47	0.51	0.37	0.46
Recall@1	0.24	0.29	0.20	0.24	0.19	0.31	0.24	0.29	0.24	0.27	0.22	0.28
F_score@1	0.30	0.37	0.23	0.29	0.22	0.33	0.29	0.37	0.30	0.33	0.26	0.33
Precision@2	0.40	0.50	0.31	0.37	0.24	0.29	0.37	0.50	0.39	0.42	0.33	0.39
Recall@2	0.42	0.53	0.35	0.43	0.36	0.44	0.43	0.53	0.39	0.43	0.39	0.47
F_score@2	0.39	0.49	0.31	0.37	0.28	0.33	0.37	0.49	0.37	0.40	0.34	0.40
Precision@3	0.35	0.42	0.28	0.31	0.21	0.23	0.31	0.42	0.34	0.37	0.29	0.33
Recall@3	0.55	0.67	0.46	0.53	0.46	0.53	0.53	0.67	0.50	0.55	0.50	0.57
F_score@3	0.41	0.49	0.33	0.37	0.28	0.31	0.37	0.49	0.38	0.42	0.34	0.39
Precision@5	0.29	0.31	0.23	0.25	0.16	0.18	0.25	0.31	0.28	0.30	0.23	0.25
Recall@5	0.75	0.80	0.62	0.68	0.59	0.64	0.68	0.80	0.67	0.71	0.66	0.71
F_score@5	0.40	0.43	0.32	0.34	0.24	0.27	0.34	0.43	0.37	0.39	0.32	0.35
Precision@10	0.18	0.18	0.15	0.15	0.10	0.11	0.15	0.18	0.18	0.18	0.15	0.15
Recall@10	0.91	0.90	0.79	0.82	0.75	0.79	0.82	0.90	0.84	0.85	0.82	0.84
F_score@10	0.29	0.29	0.24	0.24	0.18	0.19	0.24	0.29	0.28	0.29	0.24	0.25

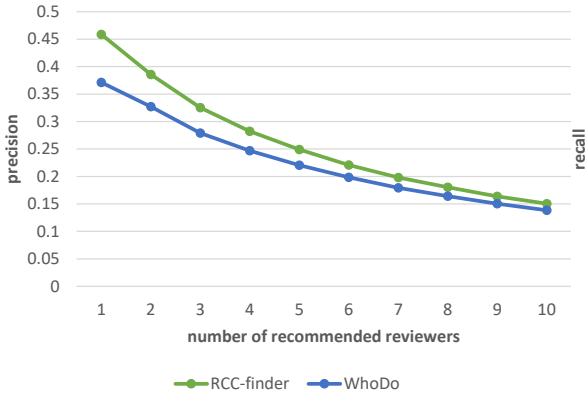


Figure 2. Comparison of the Precision@m of WhoDo with RCC-Finder for integer values of 1 to 10

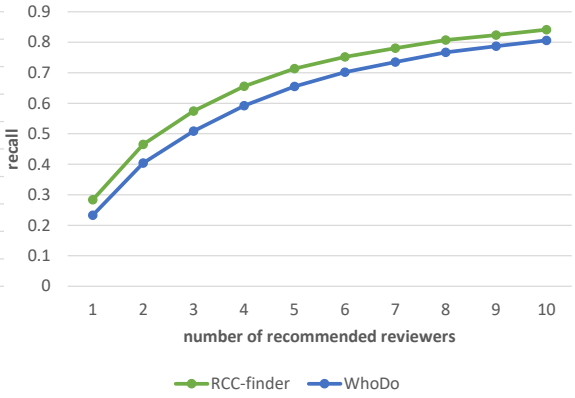


Figure 3. Comparison of the Recall@m of RCC-Finder with WhoDo for integer values of 1 to 10

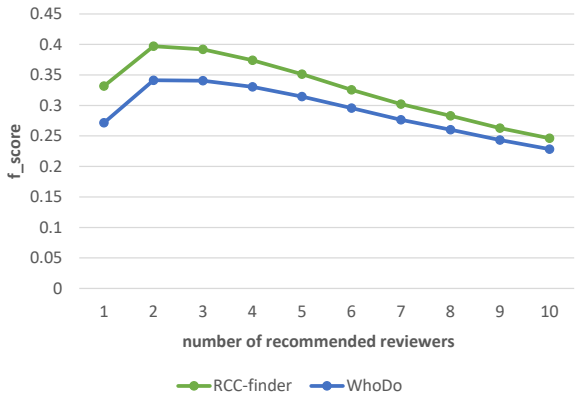


Figure 4. Comparison of the F_score@m of RCC-Finder with WhoDo for integer values of 1 to 10

from 1 to 10. Figure 2 shows the precision@m values, Figure 3 represents the recall@m values, and Figure 4 illustrates the f_score@m values for RCC-Finder compared to WhoDo.

To complete the evaluation, we calculated top-k accuracy and MRR for the two approaches along with the three criteria examined by WhoDo's developers. The results can be seen in Table 4. It shows us the quality of code reviewers' ranking in RCC-Finder in terms of MRR, providing the answer for the research question RQ2.

As visual observation transmits information better and faster, we calculated the top-k accuracy of RCC-Finder and WhoDo for all integers from 1 to 10 and provided the results in Figure 5.

Whether this improvement in the mean state reflects in the results of all projects is another question to consider. In other words, the problem is that our approach works better only in certain types of projects, which improves results in the average state or shows better performance across all projects. It's apparent from Table 4 that each project has improved, but to provide an intuitive comparison, the MRR values of RCC-Finder versus WhoDo for different projects are plotted in Figure 6.

As we know, in the WhoDo algorithm, all sentences have the same coefficient of one. While in our proposed approach, we considered the optimal coefficient for each sentence. Therefore, we have once again compared WhoDo to our approach while equalizing the

Table 4. Comparison of top-k accuracy and MRR for WhoDo and RCC-Finder

Project name	Zookeeper		Kafka		Beam		Flink		Spark		Average	
Approach	Whodo	RCC-Finder	Whodo	RCC-Finder	Whodo	RCC-Finder	Whodo	RCC-Finder	Whodo	RCC-Finder	Whodo	RCC-Finder
top-1	0.44	0.56	0.35	0.41	0.27	0.39	0.34	0.43	0.47	0.51	0.37	0.46
top-3	0.80	0.88	0.65	0.72	0.56	0.63	0.62	0.72	0.75	0.80	0.67	0.75
top-5	0.92	0.94	0.76	0.83	0.69	0.74	0.71	0.82	0.88	0.90	0.79	0.85
top-10	0.98	0.96	0.89	0.91	0.83	0.85	0.82	0.91	0.96	0.96	0.90	0.92
MRR	0.64	0.72	0.52	0.59	0.44	0.53	0.50	0.59	0.64	0.67	0.55	0.62

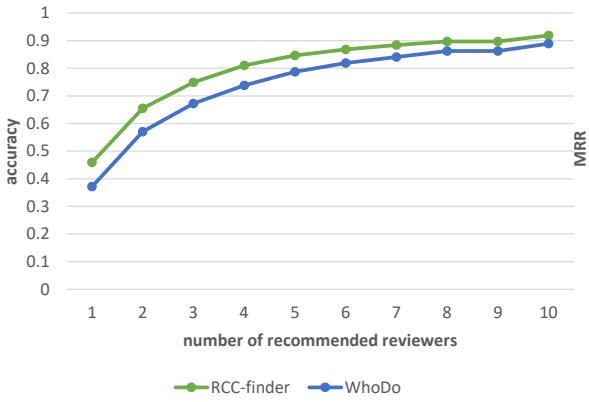


Figure 5. Comparison of the top-k accuracy of RCC-Finder with WhoDo for integer values of 1 to 10

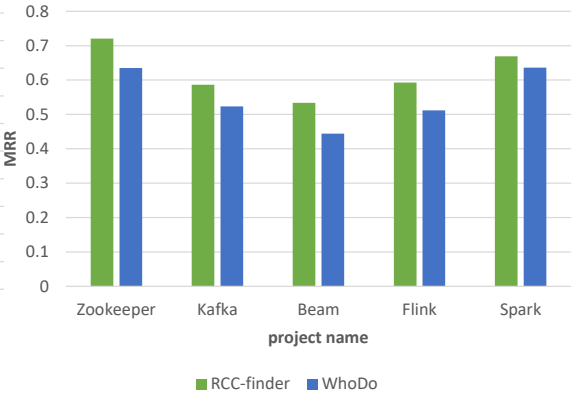


Figure 6. Comparison of the MRR of RCC-Finder with WhoDo by different projects

coefficients of all RCC-Finder sentences, so that we can assure fairness of the comparison. The results of this comparison in terms of the precision@m and recall@m for the integer values of 1 to 10 can be seen in Figure 7 and Figure 8, respectively.

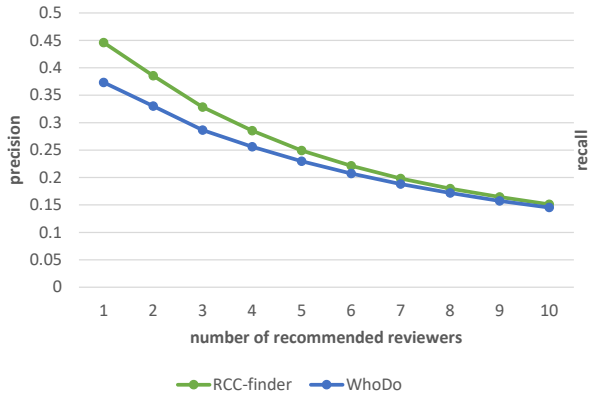


Figure 7. Comparison of the Precision@m of RCC-Finder with WhoDo with equalized coefficients for integer values of 1 to 10

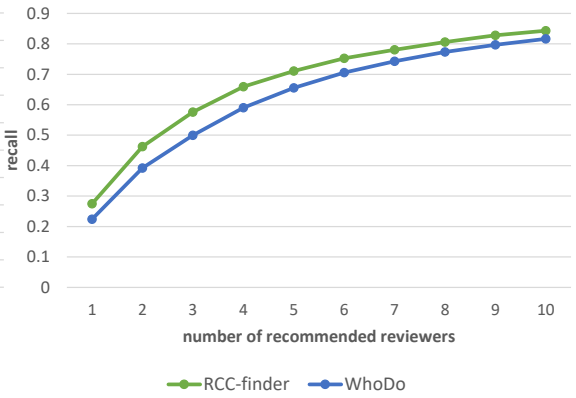


Figure 8. Comparison of the Recall@m of RCC-Finder with WhoDo with equalized coefficients for integer values of 1 to 10

3.3.2. Qualitative results

Now, according to the provided comparative charts, we can answer the research question RQ3. In all five projects, our algorithm has a higher value for almost all metrics than WhoDo. We only have slightly lower recall@10 and top-10 accuracy than WhoDo for the Zookeeper project. Therefore our approach outperforms WhoDo. As expected, our algorithm improves the automated reviewer recommendation for pull requests.

It appears from Figure 2 that the precision for the first recommendation has the highest value and has the most improvement compared to WhoDo, which is about nine percent. This amount of precision improvement is significant. Having more recommendations decreases precision since the precision formula's denominator is the recommendations number. In the numerator of this fraction, there is a subscription of recommended and actual reviewers. It's often the case that the actual reviewers of pull requests are a few, on average about two, and therefore they have little in common with the recommended reviewers. As a result, by increasing the number of reviewers recommended, the fraction denominator will grow much faster than the numerator, which leads to a decrease in precision. The precision does not improve as much in case of more recommendations since WhoDo can identify all the real reviewers for more pull requests. Therefore, the number of incorrectly answered requests in WhoDo decreases. So, the improvement we can make in these situations is less. The low precision can be partly attributed to developers not correctly selecting the actual reviewers. There are times in practice when candidates with less expertise and appropriateness are selected for review due to greater availability, individual contrasts, or to familiarize newcomers with the project. Moreover, the number of recommended reviewers in the denominator is much greater than the number of actual reviewers and their subscriptions with the recommended reviewers, which is in the numerator.

In Figure 3, the recall has the highest value for the top ten recommendations, and the top two recommendations have the most recall improvement over WhoDo, which is about eight percent. This amount of enhancement in the recall is desirable. There has been a general trend of increasing recall values and decreasing improvement with the increase in recommendations. For more suggestions, the recall value increased because the denominator of the recall formula is the number of actual reviewers, which remains constant regardless of the number of recommendations. A subscription of actual reviewers and recommended reviewers is in the numerator of this fraction, which increases with more recommendations. Therefore, as the number of recommended reviewers increases, the recall get higher due to the growth in the numerator and stability of the denominator. For more recommendations, for the same reason we have stated about precision, there has been low improvement in the recall. Also, the value of recall for more suggestions in WhoDo is very high and about 90%, so it is less likely to improve.

Figure 5 shows accuracy has the highest value for top-10 recommendations, whereas top-1 recommendations yield the highest accuracy improvement over WhoDo, which is about nine percent. An improvement of this magnitude in accuracy is highly desirable. More recommendations increase the probability of getting at least one correct answer, which is the reason for increased accuracy. There is less improvement in accuracy for more recommendations. This decrease has the same explanation as decreases in recall and precision.

As shown in the Figure 6, the MRR value and consequently the quality of our candidate rankings in all projects is better than WhoDo. A 7% improvement in this criterion in the average condition is significant.

3.4. Threats to validity

The validity of this research is threatened by four types of threats: internal, external, construct, and reliability.

- **Internal validity.** This category of threats relates to the analysis and design of the approach and to factors that affect accuracy. According to [10], there are two internal threats to the data we used. A person's different user accounts are treated as distinct individuals. In addition, a robot may sometimes make automatic changes to the files of a pull request. Here, the robot is regarded as a committer.
- **External validity.** Contains threats that hinder the generalization of results to a variety of datasets. The results of our approach were evaluated using five popular and authoritative GitHub projects. With a wide range of project sizes and programming languages used, the dataset possesses acceptable generality. However, based on the evaluation of such a limited dataset, we cannot claim that our approach will yield good results for all possible projects.
- **Construct validity.** Our approach was evaluated by considering real reviewers as the best candidates, but this assumption is not always accurate. However, as far as we know, all previous works have evaluated their approach with such an assumption. Furthermore, we sought to predict the score of each person's expertise and collaboration using the formulas we presented. Nevertheless, there is no proof that our algorithm works better than all other formulas. Furthermore, we are not certain how efficient our proposed reviewers will be in practice. However, pull request authors can scroll through the list of top ten recommendations and choose the most suitable candidates at their discretion.
- **Reliability.** The reliability of an approach determines the possibility of repeating the evaluation process on the same inputs and receiving the same output. Our algorithm has a fixed formula. Therefore, all results are definite and there are no probabilistic parts in it, so if the evaluation process is repeated with the same data, the results will surely be the same. Furthermore, all the source codes and dataset of RCC-Finder are available online at https://github.com/ISE-Research/RCC-Finder. Therefore, we do not recognize any threats to the reliability of our approach.

4. Discussion

As compared to previous studies, our proposed approach has some strengths and advantages. Our scores are according to a comprehensive set of candidate features, including review experience, commit experience, collaboration records, and activity history. Previously, studies that show a combination of all three categories, expertise, collaboration, and activity were rare. Previous researchers considered a linear coefficient of time when calculating the latency. This is the first study to examine the various nonlinear coefficients of time and to use the best case, which is the third root. Also, by determining the optimal coefficients within the formula, we quantified the relative importance of each score. Based on our algorithm, we conclude, for instance, that individuals' review history is the most effective feature among those we considered. In addition, our algorithm, based on the evaluations, also recommends suitable reviewers with the desired accuracy, outperforming the previous method. Further, our database included five GitHub projects that varied widely in both project size and programming language. This makes our findings relatively generalizable.

On the other hand, this study has some shortcomings and weaknesses. During this study, as in previous ones, the criteria used compare the proposed reviewers of our approach with the actual reviewers of the new pull request. There are times, however, when actual reviewers are not the best people to review. Furthermore, we have good results from our algorithm, yet in practice, most reviews may be carried out by a small group of expert reviewers, creating an unbalanced workload for developers. Moreover, like previous methods, we recommend each pull request suitable reviewers once upon its creation. While it is not uncommon for a request to remain open for months and the level of collaboration and expertise of the project members during this time can greatly vary. As a final note, despite the high generality of our dataset, only five projects from a single review platform cannot demonstrate the algorithm's superiority in all situations.

5. Related work

In this section, we review the related work on automated code reviewer recommendation in four categories: heuristics-based approaches, machine learning-based approaches, social network-based approaches, and hybrid approaches. In general, all the features of code reviewers used in previous works fall into one of the following four categories:

- Expertise: includes features that demonstrate the level of reviewer knowledge about new pull request changed files, such as the number of or delay of reviews or commits on these files.
- Collaboration: contains items about friendship or relationships between the reviewer and the author of the new pull request, such as the number and delay of the previous reviews on author pull requests by the reviewer.
- Activity: includes items that indicate the amount of time and effort the reviewer spends on the project as a whole, such as the total number of reviews and commits on all pull requests of the project.
- Workload: contains features that demonstrate how busy a reviewer is and so how likely it is that he or she reviews the new pull request, such as the number of his or her open review requests.

In heuristics-based approaches, historical data is used to calculate a score for each candidate reviewer through heuristical algorithms, and then the candidates are sorted by their score value to determine the most relevant reviewers. For instance, Thongtanunam et al. [1] introduced RevFinder. Its heuristic is that files in the same paths are similar and can be reviewed by the same expert code reviewers. In another work [3], Bahrami Zanjani et al. proposed an approach called CHRev in which frequency, workdays, and recency of prior code reviews are calculated. Mirsaeedi and Rigby [8] have developed Sofia, which combines CHRev's advanced reviewer recommendation engine with TurnOverRec's learning and retention recommendation engine. Sofia distributes knowledge when files under review are at risk of turnover, but suggests experts otherwise. Asthana et al. [5] proposed WhoDo. They use a heuristic algorithm to recommend reviewers based on the history of commits and reviews, which considers the reviewers' workload to reduce the impact of unbalanced recommendations. Rebai et al. [6] devised a multi-objective search algorithm to find a trade-off between expertise, availability, and collaboration history By analyzing current content and resources. Chouchen et al. [21] introduced WhoReview which finds reviewers with the most experience with code changes under review while taking account of their current workload using an Indicator-Based Evolutionary Algorithm (IBEA). Sülün

et al. in [22] calculate reviewers' scores based on the fact that an individual's knowledge of an artifact is directly correlated with the number of paths linking the individual with the artifact in a software artifact traceability graph, while inversely is proportional to the length of those paths. Then, in [23], they further developed that work by taking into account links latency. In [9] Al-Zubaidi et al. proposed a multi-objective search-oriented approach that utilized the NSGA-II algorithm to employ five criteria: code ownership, review experience, familiarity with the request author, review participation rate, and review workload. By using the NSGA-II algorithm, Chouchen et al. in [7] attempted to maximize expertise and collaboration while minimizing reviewers' workload. The previous heuristic approaches usually limited their algorithms to features from one or two categories. Especially most of them concentrated on review expertise. Also, when combining different scores through addition, all of them presumed the weight of all features equally, although their influence may differ. In our heuristic algorithm, we tried to consider features from three categories; expertise of review and commitment, collaboration, and activity. Then, we try to identify each feature's relative importance by finding the best coefficient for each sentence in the algorithm's weighted sum.

Implementing machine learning-based approaches involves building a model based on training data. Then, the performance of this model is evaluated in the prediction phase on the test data. For example, Xia et al. [24] proposed an approach called TIE that combines two learning models. One is a text-mining model that has been developed based on the textual content of the description section, the file paths, and the time of upload. In the other model, the similarities between a new review and previous reviews are calculated using a time-aware, file location-based similarity model. Sadman et al. in [25] used natural language processing techniques, a genetic algorithm, and a neural network. A variety of data is measured, including responsiveness (server logins), experience (developers' profiles and reviews previously submitted), and acquaintanceship (developers' associations with modified code blocks). Ye et al. in [26] proposed a multi-instance-based deep neural network model using CNN and LSTM networks. To recommend reviewers for pull requests, they consider three attributes, namely the title of the pull request, the commit message, and the changes made to the code. Based on a socio-technical graph, Zhang et al. in [27] proposed Coral, a new graph-based machine learning model. The graph contains a variety of entities (developers, files, pull requests, etc.) and relationships among them in modern source code management systems. They trained a graph convolutional neural network (GCN) on this graph. Chueshv et al. in [10] introduced a form of collaborative filtering, and more precisely, matrix factorization that enables the recommendation of both regular reviewers and new reviewers. In terms of precision, recall, and accuracy, most previous works in this category have delivered unsatisfactory results. Some others, like TIE, only consider features about expertise. Utilizing a broader range of feature categories, we tried to develop an algorithm that yields acceptable results.

Utilizing multiple social networks, social network-based approaches can identify the relationships between developers and their similarities, which can be used to select reviewers for new pull requests. Yu et al. [28] use cosine similarity to measure semantic similarity between pull requests based on their title and description and predict developers' scores based on how many comments they have written for similar pull requests. Moreover, they calculate collaboration scores by building a comment network between developers. Afterward, using machine learning, data retrieval, and location of files, Yu et al. [19] implemented three common approaches for assigning reviewers to pull requests. In addition, they created a Comment Network and combined it with traditional approaches. They found that hybrid

approaches' overall efficiency is more stable than using different approaches separately. As a pull request may have multiple reviewers and potential influence between them, Rong et al. [29] adapted the hypergraph technique to model these high-order relationships and introduced the HGRec. Modeling of more elements is possible with HGRec, thanks to its flexible and natural model architecture. The approaches from this category typically emphasize collaboration and sometimes expertise in reviewing and have a low recall rate. Our approach considering a wide range of feature categories, such as committing expertise, led to a higher recall rate for reviewer recommendations. It means our recommended list covers more correct answers among candidate reviewers.

A hybrid approach uses a combination of different approaches explained previously to determine who should review the new pull requests. Yang et al. [30] combined an expert-based approach with data retrieval methods available in RevFinder. With the addition of a support vector classifier, dividing reviewers into technical and managerial types, they proposed a two-tier code reviewer recommendation model called RevRec. Liao et al. [31] introduced TiRR that automatically generates the topics distribution related to each reviewer, creates a reviewer-request network and a reviewers' interest network, and determines the influence weight of reviewers by analyzing these networks. Then it calculates the probability of the new pull request topic and recommends top-k reviewers. Xia et al. [32] organized the data of previous reviews into a matrix to use collaborative filtering. To capture implicit relations between reviewers and pull requests, the authors used a hybrid approach combining latent factor modeling and neighborhood methods. Pandya et al. [33] proposed CORMS, which calculates the similarity between file paths, projects/subprojects, and author information using similarity analysis. It leverages machine learning-based predictive models for reviewer recommendations based on change topics. Assavakamhaenghan et al. [34] used ChatBots along with recommendation systems to create an interactive experience for suggestions. They suggest how using a ChatBot might improve the solution, to provide more accurate and realistic reviewer recommendations. Kong et al. [35] proposed CAMP to suit the context of proprietary software development. It considers collaboration by creating a network of all participants. Using an identifier splitting algorithm, CAMP extracts common information from pull request text and file paths to account for expertise. Some previous hybrid approaches, like RevRec, evaluated their algorithm with the data of just one or two projects, which is a small dataset, resulting in unreliable results. We tried to use more data in our approach evaluation. Some others have undesirable results in terms of accuracy, precision, or recall, like RevRec, TIRR, and [32]. Another weakness among all previous works and different approaches is their attitude to the influence of latency. Our approach to considering latency in historic records importance optimally, compares different coefficients of elapsed time, while previous works either ignored it or divided scores by a simple factor of elapsed time.

6. Conclusions and future work

The review of code changes by a third party is highly beneficial before they are incorporated into the master program. Today's distributed environments make it challenging to choose the right reviewers for new pull requests. Several studies have previously proposed automated approaches for reviewer recommendation. It is important to note, however, that most of them focus on a limited set of features of candidate reviewers. Combining different features led to better results in our experiments. As a result, we proposed in this research a heuristics-based algorithm that ranks review candidates based on their expertise level

and previous collaboration history. This effort was evaluated using the dataset presented in [10], which consists of five GitHub projects with various characteristics. In terms of top-1 accuracy, top-3 accuracy, and mean reciprocal rank, our approach achieved 46%, 75%, and 62% values, respectively.

Our future research plans are to consider other features, such as the candidates' areas of interest and the hours of the day they are usually available to review, to recommend a reviewer. Additionally, instead of recommending reviewers only once upon the creation of a pull request, it may be better to update the list of reviewers whenever a new commit or review related to the pull request files is made.

Acknowledgments

The authors did not receive support from any organizations for the submitted work.

References

[1] P. Thongtanunam, C. Tantithamthavorn, R.G. Kula, N. Yoshida, H. Iida et al., "Who should review my code? A file location-based code-reviewer recommendation approach for modern code review," in *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2015, pp. 141–150.

[2] S. McIntosh, Y. Kamei, B. Adams, and A.E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, Vol. 5, No. 21, 2016, pp. 2146–2189.

[3] M. Bahrami Zanjani, H. Kagdi, and C. Bird, "Automatically recommending peer reviewers in modern code review," *IEEE Transactions on Software Engineering*, Vol. 42, No. 6, 2016, pp. 530–543.

[4] J. Lipcak and B. Rossi, "A large-scale study on source code reviewer recommendation," in *44th Euromicro Conference on Software Engineering and Advanced Applications*, 2018, pp. 378–387.

[5] S. Asthana, R. Kumar, R. Bhagwan, C. Bird, C. Bansal et al., "WhoDo: automating reviewer suggestions at scale," in *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 937–945.

[6] S. Rebai, S. Molaei, A. Amich, M. Kessentini, and R. Kazman, "Multi-objective code reviewer recommendations: Balancing expertise, availability and collaborations," *Automated Software Engineering*, Vol. 27, No. 3, 2020, pp. 301–328.

[7] M. Chouchen, A. Ouni, M.W. Mkaouer, R.G. Kula, and K. Inoue, "Recommending peer reviewers in modern code review: A multi-objective search-based approach," in *Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 307–308.

[8] E. Mirsaeedi and P.C. Rigby, "Mitigating turnover with code review recommendation: Balancing expertise, workload, and knowledge distribution," in *42nd ACM/IEEE International Conference on Software Engineering*, 2020, pp. 1183–1195.

[9] W.H.A. Al-Zubaidi, P. Thongtanunam, H.K. Dam, C. Tantithamthavorn, and A. Ghose, "Workload-aware reviewer recommendation using a multi-objective search-based approach," in *16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering*, 2020, pp. 21–30.

[10] A. Chueshev, J. Lawall, R. Bendraou, and T. Ziadi, "Expanding the number of reviewers in open-source projects by recommending appropriate developers," in *IEEE International Conference on Software Maintenance and Evolution*, 2020, pp. 499–510.

[11] P. Thongtanunam, S. McIntosh, A.E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in thescope of modern code review," in *38th International Conference on Software Engineering*, 2016, pp. 1039–1050.

[12] A. Ouni, R.G. Kula, and K. Inoue, "Search-based peerreviewers recommendation in modern code review," in *IEEE International Conference on Software Maintenance and Evolution*, 2016, pp. 367–377.
[13] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M.W. Godfrey, "Investigating code review quality: Do people and participation matter?" in *IEEE International Conference on Software Maintenance and Evolution*, 2015, pp. 111–120.
[14] O. Baysal, O. Kononenko, R. Holmes, and M.W. Godfrey, "The influence of non-technical factors on code review," in *20th Working Conference on Reverse Engineering*, 2013, pp. 122–131.
[15] M.M. Rahman, C.K. Roy, and R.G. Kula, "Predicting usefulness of code review commentsusing textual features and developer experience," in *14th IEEE/ACM International Conference on Mining Software Repositories*, 2017, pp. 215–226.
[16] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: Examining the effects ofownership on software quality," in *19th ACM SIGSOFT symposium and the 13th European conference on Foundations of Software Engineering*, 2011, pp. 4–14.
[17] S. Ruangwan, P. Thongtanunam, A. Ihara, and K. Matsumoto, "The impact of human factors on the participation decision of reviewers in modern code review," *Empirical Software Engineering*, Vol. 24, No. 2, 2019, pp. 1016–2019.
[18] A. Bosu, J.C. Carver, C. Bird, J. Orbeck, and C. Chockley, "Process aspects and social dynamics of contemporary code review: Insights from Open Source development and industrial practice at Microsoft," *IEEE Transactions on Software Engineering*, Vol. 43, No. 1, 2016, pp. 56–75.
[19] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in GitHub: What can welearn from code review and bug assignment?" *Information and Software Technology*, Vol. 74, 2016, pp. 204–218.
[20] Y. Hu, J. Wang, J. Hou, S. Li, and Q. Wang, "Is there a "golden" rule for code reviewer recommendation? – An experimental evaluation," in *20th IEEE International Conference on Software Quality, Reliability and Security*, 2020, pp. 497–508.
[21] M. Chouchen, A. Ouni, M.W. Mkaouer, R.G. Kula, and K. Inoue, "WhoReview: A multi-objective search-based approach for code reviewers recommendation in modern code review," *Applied Soft Computing*, Vol. 100, No. 106908, 2021.
[22] E. Sülün, E. Tüzün, and U. Doğrusöz, "Reviewer recommendation using software artifact traceability graphs," in *15th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2019, pp. 66–75.
[23] E. Sülün, E. Tüzün, and U. Doğrusöz, "RSTrace+: Reviewer suggestion using software artifact traceability graphs," *Information and Software Technology*, Vol. 130, 2021.
[24] X. Xia, D. Lo, X. Wang, and X. Yang, "Who should review this change? Putting text and file location analyses together for more accurate recommendations," in *IEEE International Conference on Software Maintenance and Evolution*, 2015, pp. 261–270.
[25] N. Sadman, M.M. Ahsan, and M.A.P. Mahmud, "ADCR: An adaptive TOOL to select Appropriate Developer for Code Review based on code context," in *11th IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference*, 2020, pp. 583–591.
[26] X. Ye, Y. Zheng, W. Aljedaani, and M.W. Mkaouer, "Recommending pull request reviewers based on code changes," *Soft Computing*, Vol. 25, No. 7, 2021, pp. 5619–5632.
[27] J. Zhang, C. Maddila, R. Bairi, C. Bird, U. Raizada et al., "Using large-scale heterogeneous graph representation learning for code review recommendations at Microsoft," in *45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice*, 2023, pp. 162–172.
[28] Y. Yu, H. Wang, G. Yin, and C.X. Ling, "Reviewer recommender of pull-requests in GitHub," in *IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 609–612.
[29] G. Rong, Y. Zhang, L. Yang, F. Zhang, H. Kuang et al., "Modeling review history for reviewer recommendation: A hypergraph approach," in *44th ACM International Conference on Software Engineering*, 2022, pp. 1381–1392.

[30] C. Yang, X.h. Zhang, L.b. Zeng, Q. Fan, T. Wang et al., "RevRec: A two-layer reviewer recommendation algorithm in pull-based development mode," *Central South University*, Vol. 25, No. 5, 2018, pp. 1129–1143.

[31] L. Zhifang, W. Zexuan, W. Jinsong, Z. Yan, L. Junyi et al., "TIRR: A code reviewer recommendation algorithm with topic model and reviewer influence," in *IEEE Global Communications Conference*, 2019, pp. 1–6.

[32] Z. Xia, H. Sun, J. Jiang, X. Wang, and X. Liu, "A hybrid approach to code reviewer recommendation with collaborative filtering," in *6th International Workshop on Software Mining*, 2017, pp. 24–31.

[33] P. Pandya and S. Tiwari, "CORMS: A GitHub and gerrit based hybrid code reviewer recommendation approach for modern code review," in *30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 546–557.

[34] N. Assavakamhaenghan, R. Gaikovina, and K. Matsumoto, "Interactive chatbots for software engineering: A case study of code reviewer recommendation," in *22nd IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2021, pp. 262–266.

[35] D. Kong, Q. Chen, L. Bao, C. Sun, X. Xia et al., "Recommending code reviewers for proprietary software projects: A large scale study," in *IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2022, pp. 630–640.

e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240109, DOI: 10.37190/e-Inf240109

An *N*-Way Model Merging Approach Based on Artificial Bee Colony Algorithm

Tong Ye*, Gongzhe Qiao**

* *College of Computer and Software, Nanjing Vocational University of Industry Technology*
** *College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics*

yetong@nuaa.edu.cn, qgz@nuaa.edu.cn

Abstract

Background: In N-way model merging, model matching plays an important role. However, the N-way model matching problem has been recognized as NP-hard.
Aim: To search the optimal or near-optimal matching solution efficiently, this paper proposes an N-way model matching algorithm based on the Artificial Bee Colony (ABC) algorithm.
Method: This algorithm combines global heuristic search and local search to deal with the complexity of N-way model matching. We evaluated the proposed N-way model merging approach through case studies and we evaluated the proposed ABCMatch algorithm by comparing it with Genetic Algorithm (GA) and Elephant Herding Optimization (EHO).
Results: The experimental results show that ABCMatch can obtain more accurate model matching solutions in a shorter time, and the average model matching accuracy of ABCMatch is 2.7725% higher than GA and 1.8804% higher than EHO.
Conclusion: Results demonstrate that our method provides an effective way for software engineers to merge UML models in collaborative modeling scenarios.

Keywords: Model driven development, Tools for software researchers or practitioners, Project management

1. Introduction

Model-Driven Software Engineering (MDSE) is an important direction of Software Engineering. It refers to the systematic use of models as first-class entities throughout the software engineering life cycle [1]. Models are less bound to the underlying implementation technology and are much closer to the problem domain. They accelerate the development process of complex systems by improving the abstraction level of software development. With the increasing complexity of software systems, it is almost impossible to model complex systems by a single user. The efficiency of software development can be greatly improved by adopting the Collaborative MDSE approach [2] where multiple stakeholders manage, collaborate, and are aware of each other's work on a set of shared models. Since UML (Unified Modeling Language) class diagram [3] is one of the most commonly used models for software modeling, this paper focuses on collaborative modeling using UML class diagrams.

Generally, collaborative modeling are divided into online (real-time) collaboration and offline collaboration. Our approach is proposed to support offline collaboration where users modify their models locally and push the changes later. In the process of offline

Submitted: 7 Nov. 2023; Revised: 21 Jun. 2024; Accepted: 22 Jun. 2024; Available online: 31 Jul. 2024

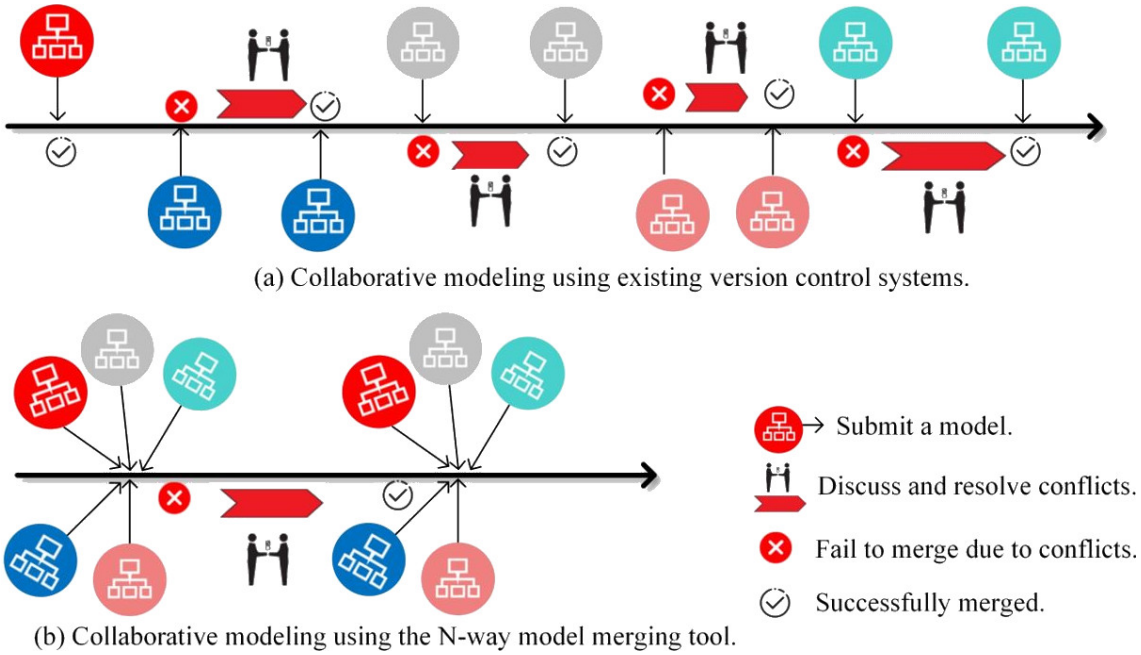


Figure 1. Comparison of collaborative modeling using existing version control systems and the N-way model merging tool

collaborative modeling, any modification will lead to different branches of the model. So it is important to merge different versions and branches periodically to obtain an integrated single model. Collaborative teams often use version control systems such as EMFStore [4] and CDO model repositor [5]. However, these tools [4, 5] only support two-way or three-way model merging. To make the motivation clear, Figure 1 gives an example of the model merging process of a five-member team, circles of each color represent the models submitted by each member. As shown in Figure 1a, using the existing version control system, each member needs to wait for others to deal with the conflicts immediately, and only after resolving the conflicts can the next merge be carried out. To save the extra waiting time, we propose a practical N-way model merging approach to merge N models at a time. As shown in Figure 1b, this approach not only reduces the number of negotiations but also saves time for submitting one by one.

In N-way model merging, the first challenge is that it is hard to well examine the overall search space effectively because N-way model matching is known as NP-hard as it requires to cope with a huge search space of possible element combinations [6]. How to efficiently check the entire search space to obtain more accurate N-way model matching solutions is a complex optimization problem. Optimization is one of the most important hot topics in scientific and technical areas [7]. Solving complex optimization problems in real life is considered to be a huge challenge. In recent years, numerous researchers have made efforts to solve optimization problems [8–11]. Some researchers use classical methods such as gradients, Lagrange, and linear mathematical to solve optimization problems [7]. However, due to the complex mathematical processes and nonlinear objective functions, classical optimization algorithms are unable to solve complex optimization problems efficiently [7]. In contrast, meta-heuristic algorithms based on group and cooperation are very effective in solving NP-hard problems [7].

Meta-heuristic methods search the optimal and near-optimal solutions by simulating natural behaviors or events [8, 12, 13]. At present, the meta-heuristic method has been listed as one of the most promising methods to solve optimization problems. Widely used meta-heuristic algorithms include Artificial Bee Colony (ABC) [14], Genetic Algorithm (GA) [15], Grey Wolf Optimizer (GWO) [16], Symbiotic Organisms Search (SOS) [17], Whale Optimization Algorithm (WOA) [18], Farmland Fertility Algorithm (FFA) [19], Elephant Herding Optimization (EHO) [20], Sparrow Search Algorithm (SSA) [21], Tunicate Swarm Algorithm (TSA) [22], Honey Badger Algorithm (HBA) [23], Northern Goshawk Optimization (NGO) [24]. Compared with traditional optimization methods, meta-heuristic algorithms have the advantages of simplicity, fewer parameters, avoiding local optimization and strong flexibility [25]. Because of these advantages, meta-heuristic methods have been widely used to solve various complex and difficult optimization problems.

Although most of the existing meta-heuristic algorithms have the above advantages, different meta-heuristic algorithms also have different weaknesses when facing different optimization issues [7]. With the change of the problem set, different optimization algorithms may show different performances [7]. Therefore, in order to solve complex optimization problems, an effective optimization method should consider all aspects of the problem. And it is necessary to select the most appropriate optimization algorithm according to the type of problem and search space [7].

The Artificial Bee Colony (ABC) algorithm [14] searches for the optimal solution through the random and objective evolution of candidate solution sets. In this algorithm, each food source represents a feasible solution to the problem to be solved, and the nectar quantity of the food source represents the fitness of the feasible solution. Bees are divided into three roles: employed bees, onlookers, and scouts. Through the cooperation of these three types of bees, the optimal solution or near-optimal solution can be obtained efficiently. ABC has good performance in searching possible solutions quickly and globally. At present, the ABC algorithm has been successfully applied in many areas including software engineering, medical image processing, economics, financial analysis, and network communication. Existing research has proved that the ABC algorithm is very suitable for solving complex and difficult combination problems [26]. Since the N-way model matching problem needs to search the model matching solution with the highest matching degree from the combinations of a large number of model elements, which is a complex combination problem, this paper selects ABC and improved it to solve the N-way model matching problem.

The second challenge is that models are complex structures connected with model relationships, so it is necessary to merge not only model elements but also their related nodes. Existing N-way model merging approaches [6, 27, 28] focus mainly on matching model elements. These approaches [6, 27, 28] ignore relationships in the model and break the chain into pieces rather than reshuffling chained elements. Unlike these methods [6, 27, 28], the proposed approach supports merging chained elements.

Conflict resolution is also an important challenge in model merging [29, 30]. Existing approaches [29, 30] transfer the responsibility of resolving conflicts to users and it is not applicable in complex merging situations where numerous conflicts lead to too many decisions to make. To prevent conflicts automatically, we present the matching model which is an intermediate form between the model matching results (generated by the proposed ABCMatch algorithm) and the merged model. Each model element in the matching model is assigned a priority number which is the same as the priority number of the model. When conflicts occur, the model element with the highest priority is picked automatically.

This paper makes the following contributions:

- We propose a new model matching algorithm ABCMatch which combines global heuristic search and local search together to deal with the complexity of N-way model matching.
- We propose a new N-way model merging approach based on the ABCMatch algorithm to merge UML class diagrams of different versions.
- We evaluated ABCMatch by comparing it with GA and EHO. The results show that ABCMatch performs better than GA and EHO in N-way model matching. The average model matching accuracy of ABCMatch is 2.7725% higher than GA and 1.8804% higher than EHO.
- We implemented the N-way model merging approach in Java and evaluated it by comparing it with EMFStore through case studies. The results show that the proposed approach performs better than EMFStore when a large number of models are required to be merged at one time in collaborative modeling.

The rest of the paper is structured as follows. Section 2 discusses related works. Section 3 presents the overview of the proposed approach. Section 4 introduces the model comparison method. Section 5 describes the ABCMatch algorithm. Section 6 presents the model merging method. Section 7 evaluates the proposed approach. Finally, Section 8 concludes this paper.

2. Related work

2.1. Two-way and three-way model merging

Model merging is a problem that has been studied for a long time in the area of collaborative modeling. In the aspect of two-way model merging, Buneman et al. propose a model merging algorithm named BDK [31], which creates the duplicate free union of two models based on the name equality of model elements. However, BDK can only identify one-type conflict as the proposed meta-meta-model contains only two relationships, *Is-a* and *Has-a*, where *Has-a* must obey one-type restriction. Pottinger and Bernstein improve BDK by defining the operator *Merge* and *take* mapping as its input [32]. However, this approach does not scale well since the manual definition of each mapping is a labor-intensive and time-consuming process.

Other studies [33–36] apply the three-way model merging technique that performs model merging on two models derived from the same ancestor model. Sharbaf et al. [33] present a novel three-way model merging approach which uses pattern-based method to detect and resolve conflicts in the merging process. Thao and Munson propose a three-way merging algorithm [34] based on LCS (Longest Common Subsequence) algorithm. Debreceni et al. propose a three-way operation-based merging algorithm [35]. However, they define only two change annotations *"must"* and *"may"*. When two changes are both annotated by *"must"* or *"may"*, the algorithm cannot determine which one to choose automatically. Our approach defines different priorities for each model thus avoiding this problem. Schwagerl et al. implement a three-way merging tool [36] for models in the Eclipse Modeling Framework (EMF). It is more general than the EMF Compare match meta-model but still fails to handle relationships as they omit the graph-like structure in the model. In this paper, we consider not only model elements but also their relationships.

2.2. *N*-way model merging

Due to the problem of selection order in two-way and three-way model merging, some approaches [6, 27, 28, 37–41] have been proposed to generate merged models from N existing variants. Schultheiÿ et al. [37] propose a heuristic N-way model matching algorithm named RaQuN, which uses multi-dimensional search trees to find suitable match candidates. Kasaei et al. [38] present a formalism for specifying N-way model merging rules. They implemented a syntax-aware editor and a parser to promote N-way merging rules for EMF-based models. Boubakir et al. [39] propose a pairwise approach for model merging, which improves the quality of the results by considering the order of combining the set of input models. Rubin et al. present the NwM algorithm [6] for the N-way merging of model variants. Assuncao et al. propose a search-based merge method [27] for UML model variants. However, these methods [6, 27] ignore relationships in the model. As models are complex structures connected with model relationships, it is necessary to merge not only model elements but also their related nodes. In this paper, we consider all input relationship chains and reshuffle elements from distinct chains by extracting the prior element link and storing it in matching models.

Jiang et al. propose an entropy-based merging tool [40] to help merge models generated by different modelers. However, it requires users to model in the way specified by the tool and cannot support merging models built with existing widely used UML modeling tools. In this paper, the proposed approach supports merging models built in the famous Papyrus modeling environment [42].

Martinez et al. present a generic framework [41] for constructing merged models from a set of model variants. But they assume that the variants are not independently generated out of the same family of models and do not target to address the problem of model similarity analysis. In addition, Reuling et al. [28] claim that Martinez et al. fail to support imprecise matching. They propose a precise N-way model merging method [28] by encoding the variability information using language-specific variability-encoding operators. However, in this method, the new class contains all duplicated class properties which require further manual handling. Our method merges duplicated model elements automatically rather than simply enumerating them.

N-way model matching plays an important role in N-way model merging. The N-way model matching problem is a complex optimization problem that requires to use optimization methods to efficiently search the optimal or near-optimal model matching solution in the huge search space. Approximate methods for solving optimization problems are divided into heuristic methods and meta-heuristic methods [43]. Heuristic algorithms usually search the optimal solution in a reasonable computing time. However, heuristic algorithms cannot guarantee the optimal solutions and are easy to fall into local optimums [43]. Due to the weaknesses of heuristic algorithms, many existing research studies have proposed meta-heuristic algorithms to solve complex optimization problems.

Meta-heuristic algorithms are inspired by natural behavior or events. The existing meta-heuristic algorithms can be divided into three categories: evolution-based algorithms, physics-based algorithms, and population-based algorithms[43]. Evolution-based algorithms mainly simulate the evolution process in nature to realize the overall progress of the population. Physics-based algorithms usually imitate physical rules to achieve optimization. Because of the strong flexibility and high performance, population-based algorithms have attracted more attention in recent years. In this type of algorithm, each population is

a biological population. Population-based algorithms search the global optimal solution through the cooperative behavior among individuals in the population.

Common population-based optimization algorithms are Artificial Bee Colony (ABC) [14], Grey Wolf Optimizer (GWO) [16], Symbiotic Organisms Search (SOS) [17], Whale Optimization Algorithm (WOA) [18], Elephant Herding Optimization (EHO) [20], Sparrow Search Algorithm (SSA) [21], Tunicate Swarm Algorithm (TSA) [22], etc. These algorithms and their variants have been widely used to find the optimal values of functions, solve multi-machine scheduling problems, multi-objective optimization problems, and so on. For example, Mohammadzadeh et al. [8] proposed the BMAMH algorithm combined with multiple swarm intelligence optimization algorithms to detect spam e-mail. Gharehchopogh [44] improved the tunicate swarm algorithm and proposed the QLGCTSA algorithm with higher performance to solve complex optimization problems. Abdollahzadeh1 et al. [45] proposed three effective binary methods based on symbiotic biological search (SOS) algorithm to solve the feature selection problem in information preprocessing. Bonab et al. 10 proposed a new hybrid method based on fruit fly algorithm (FFA) and ant optimization algorithm (ALO) to improve the performance of intrusion detection system.

Among existing population-based optimization algorithms, the ABC algorithm and its variants avoid local optimal solution by using global and local search, which has the advantages of high performance and strong flexibility. In recent years, more and more researchers choose to use the ABC algorithm and its variants to solve complex optimization problems in various fields. Öztürk et al. studied the role of the ABC algorithm and its variants in the field of medical image processing [46]. The ABC algorithm maintains a good balance between global search and local search. It can not only be used for medical image enhancement, including improving contrast, edge, artifact elimination, intelligent noise reduction, but also has played an important role in medical image segmentation, such as tumor detection, classifying image pixels into anatomical regions [46].Existing research has confirmed that the ABC algorithm has better performance than other meta-heuristic algorithms in solving various complex combination problems [26]. Because the N-way model matching problem needs to search the model combination with the best matching degree from the combinations of a large number of model elements, which is a complex combination problem, we chose the ABC algorithm and improved it to solve the N-way model matching problem.

2.3. Operation-based merging approach

Operation-based merging tries to solve the merge problem by merging operation sequences. Mansoor et al. propose an operation-based model merging method [47]. They consider merging different model versions as a multi-objective optimization problem. But the importance score of each composite option is determined by different developers. It is hard for them to compare importance scores of operations with each other while developing as they are not sure what scores others might set and this might cause their important operations disabled. In this paper, we define different priorities for input models from a global perspective thus avoiding this problem.

Some existing approaches [48–50] apply rule-based methods in operation-based model merging. Anwar et al. propose a formal approach [48] for model composition. RuCORD [49] is a rule-based composite operation detection and recovery framework for merging models in Eclipse. Chong et al. propose an operation-based approach [50] to merge different versions of UML models. However, these methods [48–50] are not applicable in large-scale projects

as they require identifying corresponding model elements and defining formal composition rules for each model element manually in the matching step. In addition, the detection and recovery processes are supposed to be guided by users which are not applicable when there are too many operations. To solve these problems, in this paper, we identify groups of corresponding input model elements automatically by the ABCMatch algorithm and handle conflicts by identifying the prior model. Furthermore, we consider N models simultaneously which is more suitable for large-scale model merging.

2.4. Conflicts resolving

Koegel et al. present an approach [29] to make conflicts part of the model and represent them as first-level entities based on issue modeling. However, this approach transfers the responsibility of resolving conflicts to users and it is not applicable in complex merging situations where numerous conflicts lead to too many decisions to make. Dam et al. propose an approach [30] to automatically resolve all inconsistencies that arise during the merging of model versions. They create a validation tree to evaluate constraint instances and build a repair tree based on the validation tree which gives repair suggestions and checks if a repair causes other inconsistencies. However, they fail to consider the situation where repair suggestions form a cycle. This method is not applicable when merging large-scale models as it may cause many cycle errors which cannot be solved automatically.

To summarize, in two-way and three-way model merging [31, 32, 34–36], results are influenced by the order to pick input model elements. Among existing N-way model merging methods [6, 27, 28, 40, 41], Rubin et al. [6], Assuncao et al. [27] and Reuling et al. [28] ignore relationships in models, Jiang et al. [40] merge UML models which are modeled in a specified way using their tool rather than common UML models, Martinez et al. [41] fail to address the problem of model similarity analysis, and Reuling et al. [28] fail to handle duplicated class properties. Among existing operation-based model merging methods [47–50], Mansoor et al. [47], fail to define the priority from a global view which might lead to important operations being disabled, and rule-based methods [48–50] require much user interaction and fail to resolve conflicts automatically. In addition, existing conflicts resolving methods [29, 30] are not applicable in N-way model merging where numerous conflicts lead to too many decisions for users to make.

To address these problems and fill the research gap, we propose an N-way model merging approach based on the ABC (Artificial Bee Colony) algorithm [14]. We identify the prior model element and prior element link to generate the merged model from the matching model. In this way, inconsistencies are avoided automatically. To solve the matching problem of N-way model merging, we propose the ABCMatch algorithm which can explore the search space and obtain the optimal matching solution efficiently.

3. Overview of the proposed approach

Existing studies [6, 48, 51] suggest that model merging can be divided into three steps: model comparison, model matching, and model combination. In the comparison step, similarity degrees between elements are calculated by comparing their corresponding sub-elements and weighing the results using empirically determined weights. These weights represent the contribution of model sub-elements to the overall similarity of their owning elements. In the matching step, pairs of elements from the input models as well as their similarity degrees

are taken as inputs, and the outputs are groups of model elements that are considered similar. In the model combination step, a new duplicate-free model that combines groups of matched elements is generated.

In this paper, we propose a novel N-way model merging approach following the above-mentioned three steps. The overview of the proposed approach is given in Figure 2.

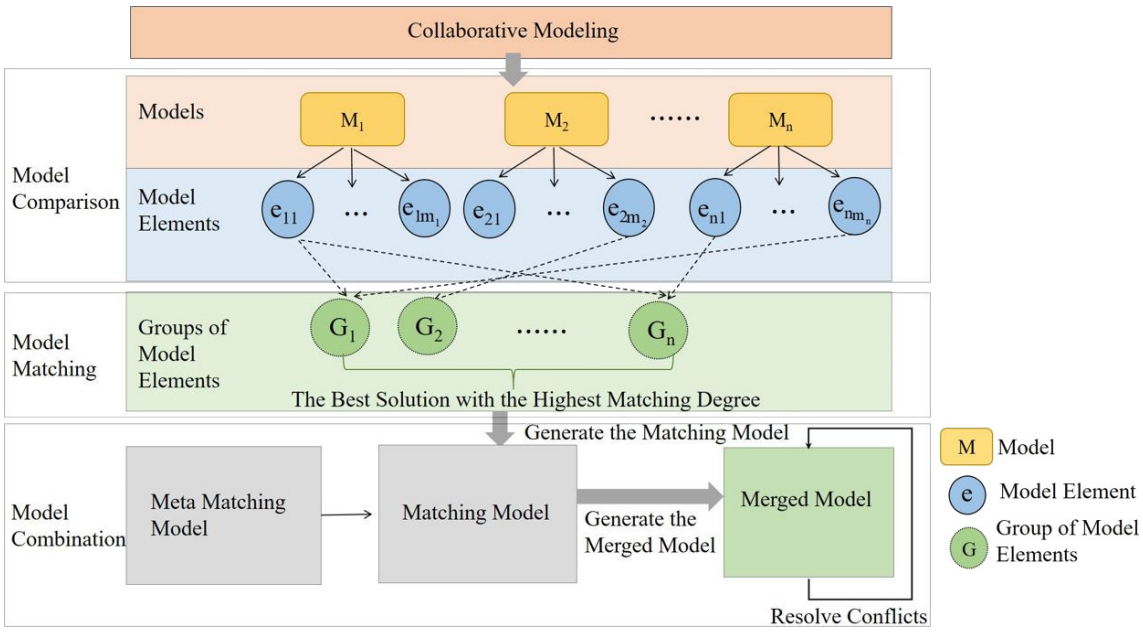


Figure 2. Overview of the proposed approach

Model comparison. The similarity degree between two classes can be calculated as a weighted sum of the similarity degrees of their names, properties, and methods. Although numerous auto or semi-auto methods have been proposed to calculate similarity degrees, gaps still exist when applying to N-way model merging. This is because, in N-way model merging, multiple input models are considered at the same time. So model comparison needs to calculate the similarity of a group of model elements rather than only two model elements in two-way or three-way model merging. To solve this problem, we propose a model comparison approach that can calculate the similarity of a group of model elements (see details in Section 4). As shown in Figure 2, models to be merged are denoted as $M_1, M_2, \ldots, M_n$. Each model M_i contains m_i model elements denoted as $e_{i1}, e_{i2}, \ldots, e_{im_n}$. We define a group of models with different versions of common elements as a matching path, which is denoted as G_i in Figure 2. Multiple matching paths without common elements constitute a model matching solution. To distinguish similarity degrees of a group of model elements and a pair of elements, the sum of similarity degrees of model elements in a matching path/solution is called the matching degree. In our approach, similarity degrees between two classes are calculated based on the Jaccard similarity coefficient [52] which is an index to measure the similarity between two sets. We calculate the matching degrees of the matching paths from two dimensions: the class name and properties/methods. Both metrics are complementary and assess two different aspects of the matching path: the first one compares the string of a group of model elements while the second one focuses on the number of properties/methods in common. A matching solution is a set of disjoint

matching paths, so the matching degree can be obtained by simply adding the matching degrees of all matching paths.

Model matching. The goal of this step is to find the optimal matching solution with the highest matching degree. The challenge is that it requires coping with a huge search space of possible element combinations [6]. To solve this problem, we propose an N-way model matching algorithm ABCMatch (see details in Section 5). First, the model matching problem is transformed into the weighted maximum matching problem of graph theory. Second, a two-dimensional integer array coding scheme of the food source is proposed by improving the food source encoding in the original ABC algorithm. Third, the strategy for generating candidate solutions is given. Then, all the feasible solutions (food sources) are exploited by employed bees, onlookers, and scouts. Finally, the best matching solution $\{G_1, G_2, \ldots, G_n\}$ with the highest matching degree is obtained which is used in the next step to generate the matching model.

Model combination. In this step, a single global merged model is generated by combining matched model elements. There exist two challenges in model combination. The first one is that existing approaches [6, 28, 40, 41] ignore relationships in the model and break the chain into pieces rather than reshuffling chained elements, while for model elements connected with relationships, it is necessary to merge not only model elements but also their related nodes. The second one is that N-way model merging is too complex for users to handle conflicts manually, conflicts should be resolved automatically. To solve these problems, we propose a novel approach to reshuffle elements from distinct chains (see details in Section 6). We present the matching model which is an intermediate form between the model matching results (generated by the ABCMatch algorithm) and the merged model. Relevant information needed for conflict resolving and structural merging is represented in the matching model. We present the meta matching model which consists of the type definitions for the objects of the matching model. As shown in Figure 2, we build a temporary matching model based on the proposed meta matching model and the best solution $\{G_1, G_2, \ldots, G_n\}$ obtained in the model matching step. Based on groups of matched model elements obtained by ABCMatch, matching model elements are generated. And for the relationships in input models, we extract the prior element links and store them in the matching model to memorize the related nodes as well as relationships between them in original models. Finally, we transform the matching model to the merged model.

4. Model comparison

The proposed matching algorithm is for UML class diagrams. We compare model elements from two dimensions: (1) name and (2) properties/methods. Assuming that the vocabulary used for naming model elements, properties, and methods are from corresponding domain terminology, then we can determine whether two elements are similar by checking if they use a similar vocabulary. This section gives the calculation method of similarity degree between any two model elements, based on which, we present equations for calculating matching degrees of matching paths and matching solutions. To make the idea more concrete, an example is given to describe the process of model comparison.

4.1. Calculation of model matching degree

Jaccard similarity coefficient [52] is an index to measure the similarity between two sets. It is widely used to compare the similarity between sample sets of limited sizes. It calculates

the similarity of sets from multiple dimensions. In each dimension, the value is usually between [0, 1]. For example, given two sets A and B, the Jaccard coefficient is defined as the ratio of the size of the intersection of A and B to the size of the union of A and B. In this paper, we use the Jaccard similarity coefficient to calculate the similarity between two model elements. For each pair of elements e_1 and e_2, the similarity degree $Si(e_1, e_2)$ is defined as the average value of the similarity degrees of their names and properties/methods. String comparison is used in the calculation of the name dimension, where the number of characters in the overlapping sub-string is divided by the total number of characters. For the property/method dimension, the similarity degree is calculated by dividing the number of common properties/methods of input elements by the number of properties/methods in the union set of these two elements.

In the following, we extend the above-mentioned calculation method for two elements to support the comparison of multiple elements in a matching path.

The matching path $d = \{e\}_d$ is composed of a set of model elements, and the matching degree $Pi(d)$ of d is the similarity among all model elements in set $\{e\}_d$. Similar to the above-mentioned method, the calculation of $Pi(d)$ also contains the same two dimensions. The first dimension of name is calculated by Equation (1), where $|\bigcap c_d|$ represents the number of characters in the overlapping sub-string of names of all model elements in $\{e\}_d$ and $\sum |c_d| - |\bigcap c_d|$ is the number of the characters in the union set of all names.

$$Similarity\_c\_mul(\{e\}_d) = \frac{|\bigcap c_d|}{\sum |c_d| - |\bigcap c_d|} \tag{1}$$

Suppose that there are N model elements in the matching path $d = e_d$. The similarity degree of properties/methods in d is calculated by Equation (2).

$$Similarity\_pm\_mul\left(\{e\}_d\right) = \frac{\sum\limits_{e_{i,j} \in \{e\}_d, i \neq j} Similarity\_pm(e_i, e_j)}{N(N-1)/2} \tag{2}$$

In Equation (2), the numerator represents the sum of the similarity degrees between any two different model elements in set $d = \{e\}_d$. Normalization of the result is implemented by dividing the total cases of taking any two model elements from N model elements in matching path d.

$$SiD(\{d\}_D) = \sum_{d_i \in \{d\}_D} Pi(d_i) \tag{3}$$

The matching solution $D = \{d_i\}(1 \leq i \leq n)$ contains multiple disjoint matching paths. As shown in Equation (3), SiD is the matching degree of D which can be obtained by simply calculating the sum of the matching degrees of all matching paths. With the help of the proposed equations, we can calculate the matching degree of any given matching path/solution.

4.2. An example of model comparison

In collaborative modeling, suppose that there are k models M_i $(1 \leq i \leq k)$, and each model M_i has m_i $(m_i \geq 1)$ model elements, which are denoted as e_{ij} $(1 \leq j \leq m_i)$. First, we calculate similarity degrees by pairs. Then, model elements from different models whose similarity degrees are larger than the predefined threshold are put into a matching

path d. Multiple matching paths without intersection constitute a matching solution D. The predefined threshold is supposed to be set by users according to actual needs. In this paper, we use $e_{ij} - e_{mn}(se)$ to indicate that the model element e_{ij} is similar to e_{mn} and the similarity degree is se. A case is given to illustrate the calculation process of model comparison.

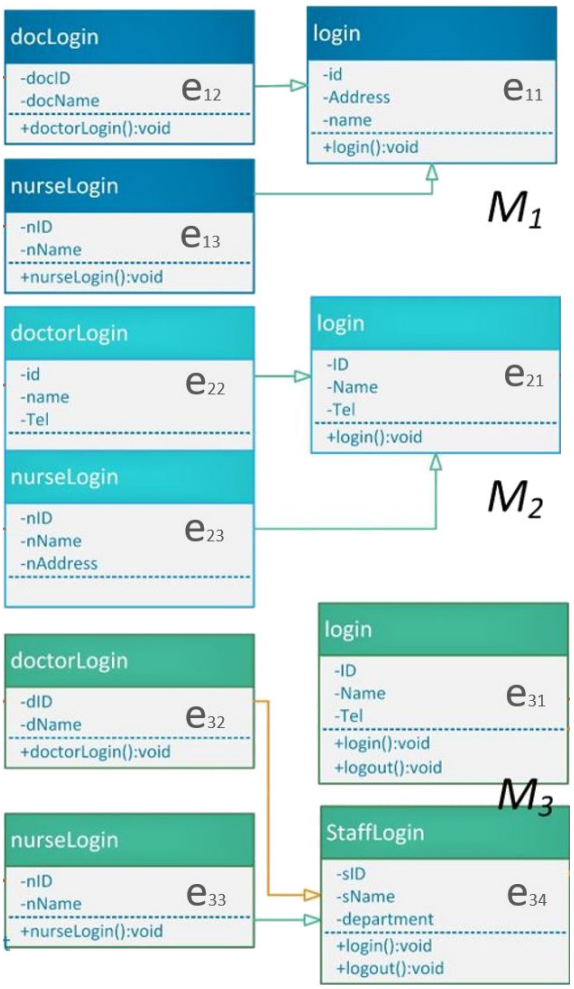


Figure 3. Three models of a "login" function in a medical system software

Figure 3 shows three models of a "login" function in a medical system software. The input models to be merged are in three colors: deep blue (M_1), light blue (M_2) and green (M_3). In the three models, the classes e_{11}, e_{21} and e_{31} describe the login function, the classes e_{12}, e_{22} and e_{32} describe the doctor login function and the classes e_{13}, e_{23} and e_{33} describe the nurse login function, which are supposed to be merged. Calculated by the proposed model comparison method, the similarity degrees of each pair of model elements are: $e_{11} - e_{21}$ (0.8750), $e_{11} - e_{22}$ (0.5983), $e_{11} - e_{31}$ (0.8333), $e_{11} - e_{32}$ (0.3125), $e_{12} - e_{21}$ (0.3846), $e_{12} - e_{22}$ (0.4211), $e_{12} - e_{31}$ (0.3846), $e_{12} - e_{32}$ (0.7179), $e_{21} - e_{31}$ (0.94), $e_{21} - e_{32}$ (0.3125), $e_{22} - e_{31}$ (0.4875), and $e_{22} - e_{32}$ (0.5).

In this example, the predefined threshold is set to 0.5. A matching solution consists of multiple matching paths without intersection. For example, $d_{a1} = \{e_{11}, e_{21}, e_{31}\}$, $d_{a2} = \{e_{12}, e_{22}, e_{32}\}$ and $d_{a3} = \{e_{13}, e_{23}, e_{33}\}$ are three valid matching paths. The three matching

paths form a matching solutions $D = \{d_{a1}, d_{a2}, d_{a3}\}$. According to Equation (1) and Equation (2), the the model matching degrees of three follows: $Pi(d_{a1}) = 0.8462$, $Pi(d_{a2}) = 0.6875$, $Pi(d_{a3}) = 0.9444$. Substituting $Pi(d_{a1})$, $Pi(d_{a2})$, and $Pi(d_{a3})$ into Equation (3), we can obtain the model matching degree of D is $SiD(D) = 2.4781$. The details of the generation of matching paths and matching solutions are described in Section 5.

5. The ABCMatch algorithm

In N-way model matching, it requires coping with a huge search space of possible element combinations to find the optimal matching solution. In this section, first, the model matching problem is transformed into the weighted maximum matching problem of graph theory. Second, a search-based matching approach based on the ABC algorithm [14] is proposed. Then, a two-dimensional integer array coding scheme of the food source is proposed by improving the food source encoding in the original ABC algorithm [14] and the strategy for generating candidate solutions is given. Finally, all the feasible solutions are exploited by employed bees, onlookers, and scouts. By searching for the best model matching solution through the bee colony's exploration of food sources, this approach can find the optimal matching solution with high efficiency.

5.1. The problem of model matching

In an undirected graph G, the points covered by an edge are defined as the endpoints of the edge. The maximum matching problem is to find the largest edge set S that contains the most edges where any endpoint in this graph is covered by only one edge. For weighted graphs, the maximum matching problem is to find an edge set S with the maximum sum of weights.

In this paper, the model elements to be matched are regarded as endpoints in the graph, the matching path containing multiple elements is regarded as the edge of the graph. And the matching degree of the matching path is regarded as the weight of the edge. The goal is to find the optimal matching solution with the highest matching degree. Suppose that $G = (V, E)$ is an undirected graph and endpoint set $V = V_1 \cup V_2 \cup \ldots \cup V_n$ is composed of n disjoint subsets, where each subset V_i $(1 \leq i \leq n)$ denotes the set of model elements in the i-th model M_i. The edge $E(i, j)$ represents the matching correspondence between element e_i of M_i and model element e_j of M_j. Assuming that there is an edge E_{ik} between e_k and e_i, and e_k does not belong to model M_i or M_j, then the edges $E(i, j)$ and $E(i, k)$ can form a matching path $E(i, j, k)$. The goal to search for the best matching solution is to find a group of matching paths with the maximum weight sum, where endpoints of each path do not intersect with each other.

Suppose that there are k models and the i-th model M_i contains m_i model elements. If no more than one model element is selected from each model to participate in model matching, there will be a total of $\prod_{i=1}^{k}(m_i + 1)$ matching paths. After excluding the situation of empty set or the situation of only one model element, the number of paths p reduces to $\prod_{i=1}^{k}(m_i + 1) - 1 - \sum_{i=1}^{k} m_i$. The set of matching solutions is the power set of all paths

minus the empty set, so there are $2^p - 1$ matching solutions. The optimal solution cannot be obtained in linear time using enumeration methods and it is easy to miss the global optimal solution using greedy algorithms. This paper improved the original ABC (Artificial Bee Colony) algorithm [14] and proposes a search-based N-way model matching algorithm ABCMatch.

5.2. Encoding

In the original ABC algorithm [14], each food source represents a feasible solution of the problem to be solved, and the nectar quantity of the food source represents the fitness of the feasible solution. Bees are divided into three roles: employed bees, onlookers, and scouts. Through the cooperation of these three types of bees, the optimal solution or approximate optimal solution is obtained with high efficiency. In this section, we propose the ABCMatch algorithm improved by the ABC algorithm to solve the N-way model matching problem. In the ABCMatch algorithm, the corresponding relationship between bee colony foraging behavior and model matching problem is given in Table 1.

Table 1. Corresponding relationship between foraging behavior of bee colony and model matching problem

Bees foraging	Model matching
Food source position	Model matching solution
Nectar quality	Model matching degrees
Speed of searching and foraging	Speed of algorithm optimization
The best food source	The optimal model matching solution
Dimension of food source	The first dimension represents matching path The second dimension represents matching solution

In the original ABC algorithm [14], the food source position represents the feasible solution to the optimization problem, which is denoted by a multidimensional vector. In the model matching problem, each candidate solution represents a feasible model matching solution. Therefore, it is necessary to improve the food source encoding and the strategy for generating candidate solutions. Here, we use a two-dimensional integer array coding scheme to code the model matching solution. A matching path is represented by a two-dimensional array d_i, where $d[i][j] = 1$ indicates that the model element e_{ij} is in the matching path, and $d[i][j] = 0$ indicates that it is not in the matching path. Multiple disjoint matching paths compose a matching solution.

5.2.1. Initialization

The number of food sources is denoted as SN. In the initialization phase of the ABC--Match algorithm, SN feasible solutions are generated randomly. Each matching path is a two-dimensional matrix with m rows and m columns, where m is the maximum number of model elements. According to the proposed coding scheme, to select the j-th element from model M_i, we need to set the value of the element in the j-th row and i-th column to "1". If all values in a column are set to "0", which means no model element of the corresponding model is taken.

Next, we check whether all the elements in the matrix are "0" or if there is only one "1" in the matrix. If so, it means that the matching solution is empty or there is only

one model element. In that case, a new matching path is generated. In our approach, a threshold *GSiD* of matching degrees is defined, and the matching degrees of S matching paths are calculated. If the matching degree is less than *GSiD*, a new matching path will be generated until matching degrees of S matching paths are all over *GSiD*. After generating S valid paths, we compose them to generate SN matching solutions. This process needs to satisfy the following rules:

1. Each matching solution has at least one matching path.
2. Matching paths must not intersect.

In the process of path combination, if the rules are not satisfied, a new combination will be generated until SN feasible solutions are generated.

According to Equation (3), the model matching degree $SiD(D_i)$ of each matching solution D_i is calculated as the fitness value of the feasible solution. Among the fitness values of SN feasible solutions, the maximum fitness value max_v and the optimal matching solution $best\_s$ are recorded.

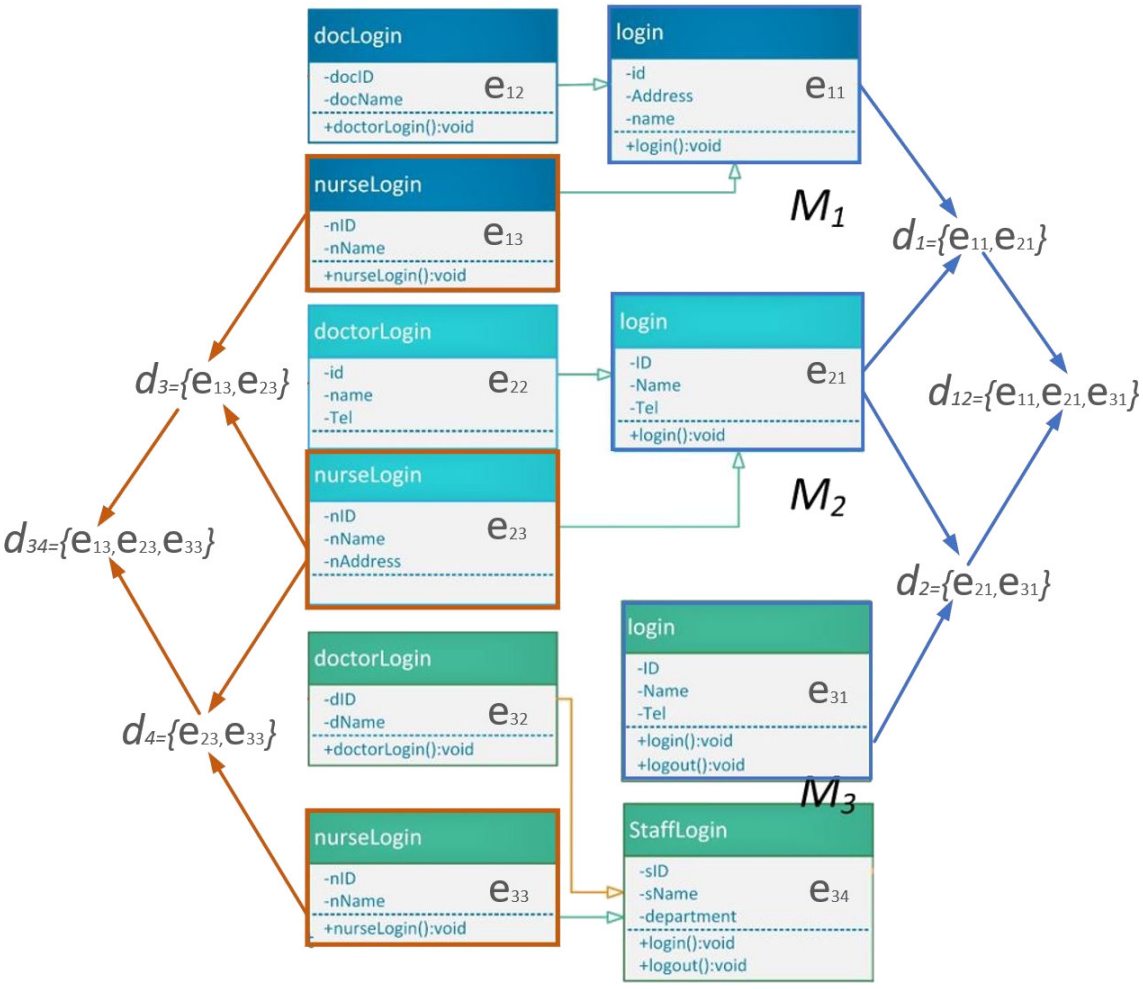


Figure 4. Matching paths in the example

An example of the matching paths is shown in Figure 4. It shows three models of a "login" function in a medical system software. The input models to be merged are in three colors: deep blue (M_1), light blue (M_2) and green (M_3). Matching paths are

a set of elements (class diagrams) from different models. For example, the matching path $d_1 = \{e_{11}, e_{21}\}$ means "login: M_1" and "login:M_2" are matched. Matching paths need to be merged if they have intersections. For example, in Figure 4, the matching paths d_1 and d_2 have intersection, which is consistent with (5), so they are merged to a new path d_{12}. Similarly, d_3 and d_4 are merged to a new path d_{34}.

In the the initialization phase, first, valid paths $d_1 = \{e_{11}, e_{21}\}$, $d_2 = \{e_{11}, e_{31}\}$, $d_3 = \{e_{12}, e_{22}, e_{32}\}$, $d_4 = \{e_{11}, e_{22}\}$, $d_5 = \{e_{13}, e_{23}, e_{33}\}$ are generated randomly. Then, five feasible matching solutions are obtained by composing the five paths $d_1, d_2, \ldots, d_5$. After deleting solutions with intersecting paths, the matching degrees of all matching solutions are calculated, and the results are shown in Table 2.

Table 2. Matching solutions and model matching degrees

No.	Matching solution	Matching degree
1	$D_1 = \{d_1, d_3\}$	1.5625
2	$D_2 = \{d_2, d_5\}$	1.7777
3	$D_3 = \{d_3, d_4\}$	1.2857
4	$D_4 = \{d_1, d_4\}$	1.4732
5	$D_5 = \{d_2, d_3\}$	1.5208

5.3. Iteration process

After initialization, all the feasible solutions (food sources) will be exploited. Each cycle includes the behaviors of employed bees, onlookers, and scouts.

5.3.1. Employed bees phase

We assign an employed bee to each food source, thus the number of the employed bees is SN too. In the initial matching solution, there are only S matching paths. The employed bee exploits the neighborhood of the food source and adds matching paths. Because adding a new matching path may cause conflicts with existing matching paths, it is necessary to determine whether to delete the conflict elements in existing paths and add the new path or just stay unchanged. In our approach, the decision is made according to the matching degree. At the beginning of the cycle, each matching path d_y in the food source is matched with each path d_x in solution D_i and the conflict rate between them is calculated. Suppose that there are k models where each model M_i has m_i model elements, the conflict rate is denoted as P, and the boolean variable m is used to indicate whether the composition of paths is necessary. There are three situations as follows:

1. If the collision rate of matching paths is 0 which means that all paths are disjoint, then there is no need to merge the paths, which is formally expressed as (4).
$$\forall i \in [1,k], \forall j \in [1,m_i], d_x[i][j] + d_y[i][j] \leq 1 \rightarrow P = 0, M = 0 \tag{4}$$
2. If two paths have only one common element and there is no other model elements from the same model in these paths, then the conflict rate is 0 and the composition of paths is necessary. It is formally expressed as (5).
$$\begin{aligned} &\forall i, p, q \in [1,k], \forall j, t \in [1,m_i], \exists d_x[i][j] + d_y[i][j] > 1 \\ &\wedge\, j \neq t \wedge d_x[p][t] + d_y[q][t] \leq 1 \rightarrow P = 0, M = 1 \end{aligned} \tag{5}$$

3. If there exist elements in the intersecting path from the same model, a conflict occurs. It is formally expressed as (6).

$$\begin{aligned} &\forall i,p,q \in [1,k], \forall j,t \in [1,m_i], \exists d_x[i][j] + d_y[i][j] > 1 \\ &\wedge j \neq t \wedge d_x[p][t] + d_y[q][t] > 1 \rightarrow P > 0 \end{aligned} \tag{6}$$

The computation method of the conflict rate P is as (7).

$$P = \frac{n}{(m_i + m_k) \times (m_i + m_k - 1)/2} \tag{7}$$

In (7), n is the number of conflict element pairs. The conflict rate is the ratio of the number of conflict pairs to the number of possible cases of taking any pair of elements in the two matching paths.

In this approach, we add the new path according to the above three situations. If $P = 0$, $M = 0$, the path will be directly added to the matching solution; If $P = 0$, $M = 1$, the path will be merged into the path that intersects with it in the matching solution. If $P > 0$, we select the matching path d_y whose conflict rate P is less than the predefined threshold P', and add the path d_y to the matching solution D_i to generate matching solution *Neighbour_*D_i. Then, we delete the model elements in the original solution that conflict with the new path. In the above-mentioned example, the paths of D_2 are re-matched. After generating new matching paths according to the rules, a new matching solution D'_2 is obtained. A new matching solution *Neighbour_*D_i is a new food source. If the model matching degree of *Neighbour_*D_i is greater than that of D_i, the food source will be updated.

In the above example, for the first employed bee, it selects food source D_1 and adds it to D_2. Then, the path in D_1 with the least matching conflict to D_2 is added to solution D_2, and the new food source is $[d_1, d_2, d_3, d_5]$. As d_1 and d_2 have intersection, which is consistent with (5), they are merged to a new path d'_{12} as shown in Figure 4. After merging, we can get the new matching solution $D'_2 = \{d'_{12}, d_3, d_5\}$. Then we calculate the matching degree $SiD(D'_2) = 2.4781$, which is better than the original $D_2 = 1.7777$, so we accept the new food source. When all employed bees have completed the search, the new population is shown in Table 3. The number updated in this iteration is indicated by the underline.

Table 3. Updated matching solutions and matching degrees

No.	Matching solution	Matching degree
1	$D_1 = \{d_1, d_3\}$	1.5625
2	$D'_2 = \{d'_{12}, d_3, d_5\}$	<u>2.4781</u>
3	$D'_3 = \{d'_3, d_4\}$	**1.5382**
4	$D_4 = \{d_1, d_4\}$	1.4732
5	$D_5 = \{d_2, d_3\}$	1.5208

5.3.2. Onlookers phase

Each onlooker selects a food source according to the probability which is proportional to the nectar quality. The selecting probability P_i is calculated by (8) [53].

$$P_i = \frac{0.9 \times \text{fit}(D_i)}{\max_{i=1}^{SN} \text{fit}(D_i)} + 0.1 \tag{8}$$

In (8), P_i is the fitness value of solution D_i, and in this paper, fit calculates the matching degree *SiD*. Onlookers select the food source by the roulette mechanism. First, it generates a random number. If the number is greater than the random number, then the onlooker will not move. Otherwise, the onlooker will attach itself to the food source and exploit its neighborhood. The greedy selection strategy is used to update the food source. Obviously, according to the selection method, the food source with higher fitness will attract more onlookers.

In the above example, the selecting probabilities are 1, 0.95, 0.87, 0,43, 0.1. For the third onlooker, first, a random number is generated, which is smaller than P_3, and then this onlooker will exploit the solution. And the new solution is $D'_3 = \{d_2, d'_5\}$. The new matching degree $SiD = 1.79 > 1.52$, thus the matching solution will be updated. For the fifth onlooker, the random number is greater than 0.1, so it does not move. After all the onlookers finish the search, the new population is shown in Table 3, and the updated number is indicated in bold.

5.3.3. Scouts phase

If a solution is not updated after limited iterations (set to 5 in the proposed algorithm), then this food source will be abandoned. The associated employed bee will become a scout and randomly generate a new food source. For the above example, after the employed bees and onlookers finish the search, the trial is [1, 3, 3, 4, 1], which did not reach the maximum value limit, thus the scout will not appear. After all the food sources are explored, the best matching solution *best_s* and the optimal matching degree *max_v* will be updated, and the next iteration will begin. Until now, *max_v* is 2.4781, and the corresponding *best_s* is $\{d'_{12}, d_3, d_5\}$. In this matching solution, the following three sets of classes are matched: $\{e_{11}, e_{21}, e_{31}\}$, $\{e_{12}, e_{22}, e_{32}\}$, and $\{e_{13}, e_{23}, e_{33}\}$.

6. Model combination

This section defines the matching model and introduces the approach of transforming the matching model to the final merged model. First, the meta-model is presented. Second, an example is given on how to build a matching model from the matching elements obtained in the model matching step. Then, we introduce how to generate the final merged model from the matching model. Finally, we prove that our approach satisfies the Generic Merge Requirements [32].

6.1. Meta-model

The proposed meta-model is given in Figure 5. It consists of the following type definitions for the objects in the matching model.

1. ***Matching element.*** *Matching elements* are the first-class objects in a matching model. Each *matching element* consists of four properties: the *Input_Model_Element_-*

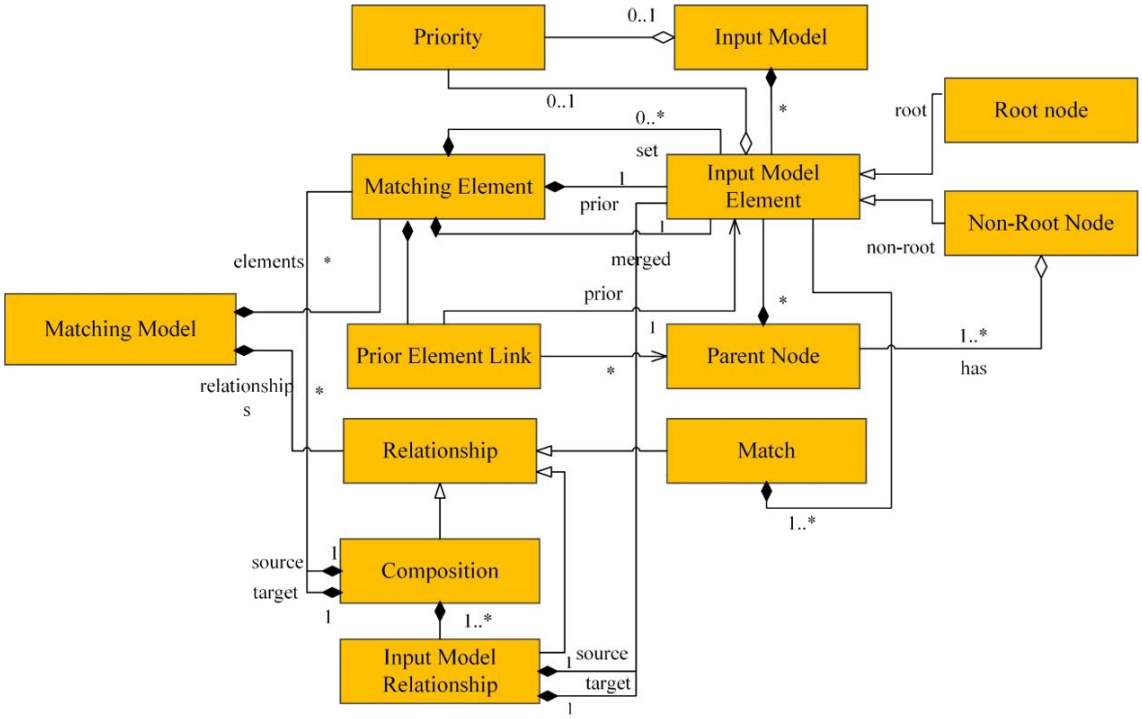


Figure 5. The proposed meta-model of the matching model

Set, the *Prior_Input_Model_Element*, the *Merged_Input_Model_Element* and the *Prior_Element_Link*.

The *Input_Model_Element_Set* is a set of matched model elements obtained by the proposed ABCMatch algorithm. By merging the model elements in an *Input_Model_-Element_Set*, the *Merged_Input_Model_Element* is generated. Each input model to be merged is given a different priority number. When conflicts occur during model merging, the model element with the highest priority is picked, and this element is called the *Prior_Input_Model_Element*. By searching the direct and indirect parent nodes until finding a root node of the *Prior_Input_Model_Element*, the *Prior_Element_Link* is obtained, which is used to build relationships of the *Merged_Input_Model_Elements* in the matching model.

2. ***Relationship.*** There are three types of relationships in the matching model: *Match*, *Composition* and *Input_Model_Relationship*. The *Match* is the relationship among multiple model elements, it represents correspondences among similar/common model elements from different models. We define *Composition* as the binary relationship between matching elements which indicates that a matching element consists of another matching element. For example, suppose that MA and MB are two matching elements where A and B are *Merged_Input_Model_Elements* in MA and MB, respectively. If B is the subclass of A, then a *Composition* relationship should be built from MB to MA. As matching is not only between the model elements themselves but also their related model elements. In addition to source and target matching elements, the relationship *Composition* also has a property that records relationship types between input model elements. The *Input_Model_Relationships* are binary relationships between input model elements, they are defined by the meta-model of input models. In this paper, input

models to be merged are UML class diagrams, so *Input_Model_Relationships* are UML relationships such as generalization, association, etc.

6.2. The model matching process

We define two operators for model matching. The operator *match* is to find a set of matched model elements from multiple models built by different collaborators and generate matching elements. The operator *connect* is to build relationships between matching elements and model elements. After operating *match* and *connect* in sequential order on a group of input models, a matching model is generated. In our approach, the models are merged based on the matching models. The inputs of *match* are as follows:

1. Input models: $M_1, M_2, \ldots, M_n$.
2. Priority of input models: $Pr_1, Pr_2, \ldots, Pr_n$.

Each model element has a priority number the same as the model it belongs to. When conflicts occur, the model element with the highest priority is picked.

The semantics of *match* is defined as follows:
The function $match((M_1, Pr_1), (M_2, Pr_2), \ldots, (M_n, Pr_n)) \longrightarrow ME$ matches n models $M_1, M_2, \ldots, M_n$ based on similarity, the detailed description of similarity calculation is given in Section 4. The function consists of three steps as follows. First, it searches for the optimal matching solution which contains multiple matching paths of similar model elements. Second, it generates a merged element for each matching path. When facing representation conflicts, the representation of the element with the highest priority is picked. Third, it records the related elements and relationships of the prior element. Finally, a matching model element ME is generated.

The semantics of *connect* is defined as follows:
The function $connect(ME_1, ME_2, \ldots, ME_n) \longrightarrow GM$ connects n matching elements ME_1, $ME_2, \ldots, ME_n$ based on original model relationships and generates a matching model GM. Relationships are added according to three different situations. (1) For each matching element ME_i $(1 \leq i \leq n)$, we search its *Prior_Element_Link*. If the model element in the link does not exist in GM, then we add it to GM as well as its relationships. (2) If the model element already exists in a matching element ME_j $(1 \leq j \leq n, i \neq j)$ in GM and there is no relationship between ME_i and ME_j, then the relationship *Composition* is added from ME_j to ME_i. And its property *Input_Model_Relationship* is set the same as the relationship in the *Prior_Element_Link*. (3) If the added model element or the matching element has related nodes, then we search its element link and repeat the steps in the first two situations.

6.3. An example of the matching model

An example of the matching model is shown in Figure 6. It shows three models of a "login" function in a medical system software. The input models to be merged are in three colors: deep blue (M_1), light blue (M_2) and green (M_3). The matching elements generated are in red and the *Merged_Input_Model_Elements* are in blue. We define that the priority order of these models is $M_3 > M_2 > M_1$.

As shown in Figure 6, matching elements are generated based on these matching paths. Take the the matching path $d_1 = \{e_{11}, e_{21}, e_{31}\}$ as an example, "Matching element *login*" is generated which consists of three properties, the set {login: M_1, login: M_2, login: M_3}, the prior element "login: M_3", and the merged element "Merged_Input_Model_Element *login*". As "login" is the root node which does not have a parent node in this example, there is no

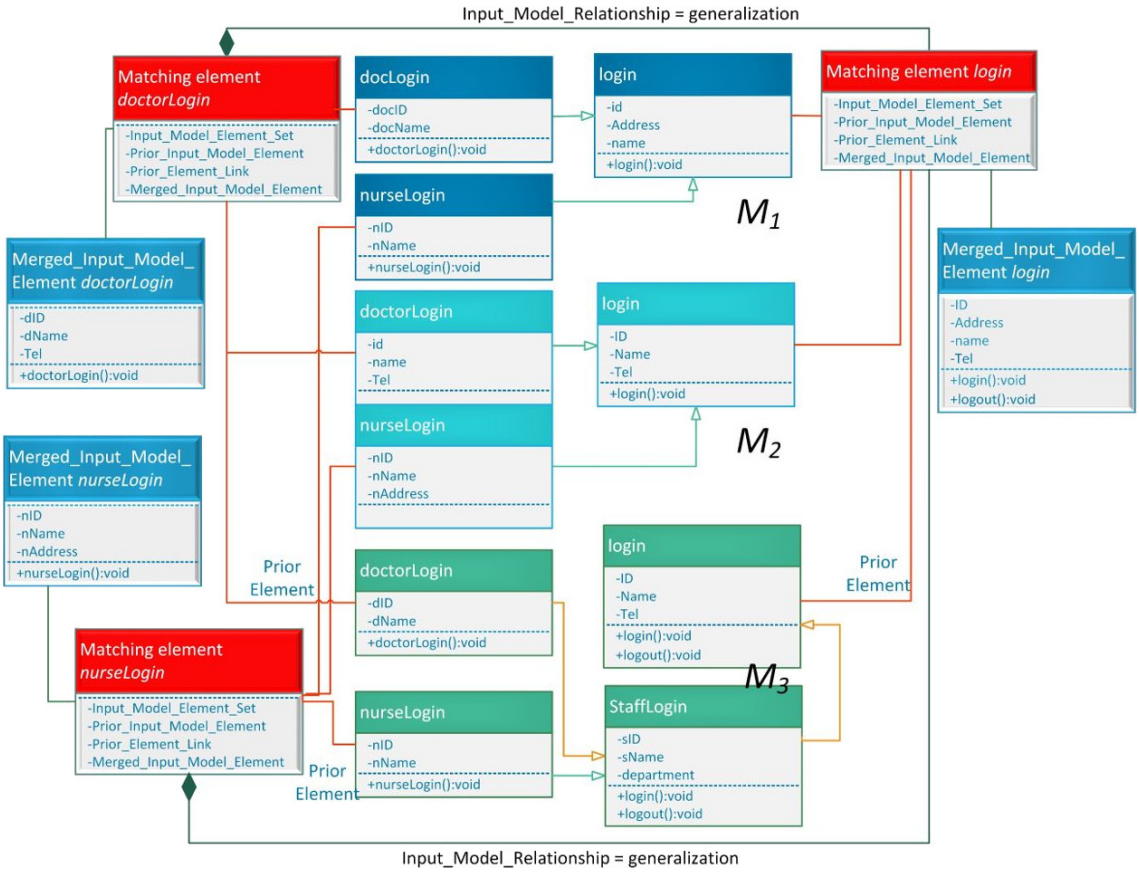


Figure 6. An example of the matching model

Prior_Element_Link in this matching element. "Merged_Input_Model_Element *login*" contains all properties and methods of the original models. It is a new "duplicate-free" model element obtained by merging "login: M_1", "login: M_2", and "login: M_3".

Models have relationships, so it is necessary to compare matched elements' related nodes which are connected by model relationships. Searching all related nodes of model elements is time-consuming. To improve the efficiency of this process, we search the prior element link of each merged element from itself to the root node and compare these nodes to other matching elements' merged elements. First, we find out all groups of matched elements and generate a merged element for each group. Second, the prior element and prior element link is memorized in each group where the merged element shares the same link with the prior element. Then, matching elements are generated and relationships between merged elements are built with the help of the *Prior_Element_Links*. For example, in Figure 6, "doctorLogin: M_3" is the prior element of "Merged_Input_Model_Element *login*". And its *Prior_Element_Link* is highlighted in orange in Figure 6. As "login: M_3" in the *Prior_Element_Link* exists in the *Input_Model_Element_Set* of "Matching element *login*", so it is replaced by "Merged_Input_Model_Element *login*".

6.4. Model merging

We merge model elements and generate a new global model M' by operating on the matching model. First, model elements that are not in *Prior_Element_Links* are deleted.

For example, "docLogin: M_1" and "login:M_1" are deleted while "StaffLogin: M_3" is reserved as it is in the *Prior_Element_Link* of "Matching element *doctorLogin*".

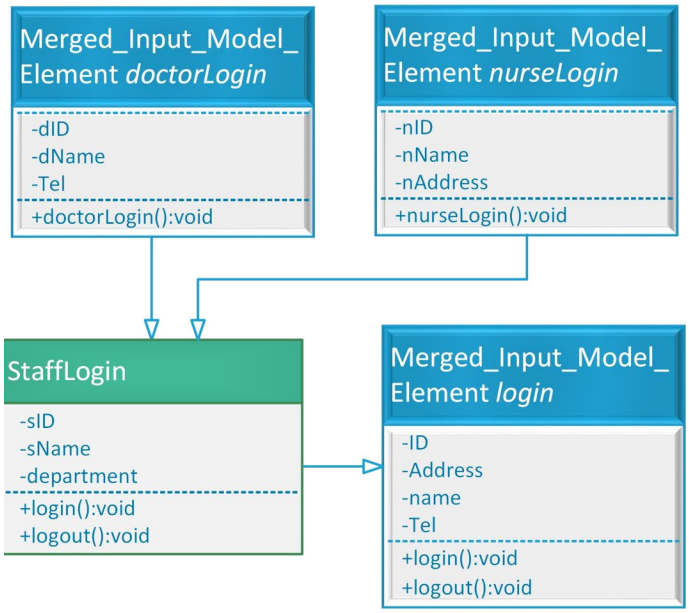


Figure 7. The merged model M' in the example

Second, merged elements are connected to model elements in *Prior_Element_Links*. For example, in the *Prior_Element_Link* of "Matching element *doctorLogin*", there is a generalization relationship which connects "doctorLogin: M_3" with "StaffLogin: M_3", so we build the same relationship between "Merged_Input_Model_Element *doctorLogin*" and "StaffLogin: M_3". Note that in this step, if the model element in the *Prior_Element_Link* exists in any matching element's model element set, it means there is a merged element of this model element. In that case, we replace this model element with the corresponding merged element. For example, "Login: M_3" is replaced by "Merged_Input_Model_Element *login*". Finally, input model elements that do not match with other elements are added to the merged model directly. In this example, the final merged model M' is given in Figure 7.

The proposed model merging method satisfies the following Generic Merge Requirements [32]. Signals used are given in Table 4.

1. ***Element preservation.*** Each element of source models has a corresponding element in the target model. Formally, for each element $e_{ij} \in M_i (1 \leq i \leq n, j > 0)$, if it has one or more matching elements in other input models, they will be matched and generate a merged model element MME_k $(k > 0)$ in the matching model ME_i, which will be added into the merged model M'. So e_{ij} must have a corresponding element $e' = MME_j \in M'$. If e_{ij} does not have any matching element but it exists in the prior element link PEL_q $(q > 0)$, then it is added to M' and connected with the merged element MME_q or model elements in $PELq$. If e_{ij} does not have any matching element and does not exist in any prior element link, it will also be added to M' with its related model elements and relations in the source model. So we can conclude that for any element $e_{ij} \in M_i$ $(1 \leq i \leq n, j > 0)$, there is always a corresponding element $e'_{ij} \in M'$.
2. ***Equality preservation.*** Input elements of source models are mapped to the same element in merged model M' if and only if they are equal in the mapping, where equality

Table 4. Signals illustration

Signal	Meaning
n	The number of input models.
M_i	The i-th input model ($1 \leq i \leq n$).
Pr_i	The priority number of model M_i.
e_{ij}	The j-th model elements in $M_i (1 \leq i \leq n)$.
M'	The global model after model merging.
GM	The matching model.
ME_i	The i-th matching element in $GM (1 \leq i \leq n)$.
MES_i	The set of model elements in matching element ME_i.
PE_i	The prior model element in matching element ME_i.
MME_i	The merged model element in matching element ME_i.
PEL_i	The prior element link in matching element ME_i.
$M(e, e')$	The input model element e has a unique corresponding element e' in M'.
$Eq(e_i, e_j)$	Input model elements e_i and e_j are equal in model merging.
$R(e_i, e_j)$	The relationship between model elements e_i and e_j.
$V(e, p)$	The value of the property p in the model element e.

in the mapping is transitive. Formally, suppose that $e_i \in M_i$, $e_j \in M_j$ and $Eq(e_i, e_j)$, which means that e_i and e_j are equal in the process of generating a matching element ME_i. Then $e_i, e_j \in MES_i$. And $M(e_i, e')$, $M(e_j, e')$ where $e' = MME_i \in M'$. Suppose that $Eq(e_i, e_j)$, $Eq(e_i, e_k)$, then $e_i, e_j \in MES_1$, $M(e_i, e'_1)$, $M(e_j, e'_1)$ and $e_i, e_k \in MES_2$, $M(e_i, e'_2)$, $M(e_k, e'_2)$. As e_i's corresponding element e' is unique in M', $e'_1 = e'_2 = e'$. So $M(e_j, e')$, $M(e_k, e')$ which means that equality in the mapping is transitive. If e_i and e_j are not equal in the mapping, then there is no such e', so that e_i and e_j correspond to different element in M'.

3. ***Relationship preservation.*** Each input relationship is explicitly in or implied by target model M'. Formally, for each relationship $R(e_{ik}, e_{ij})$ between e_{ik} and e_{ij} in M_i, there are two cases. If one or both of e_{ik} and e_{ij} have matching elements such that $e_{ik} \in MES_i$ or $e_{jk} \in MES_j$, then $R(e_{ik}, e_{ij})$ is recorded in PEL_i or PEL_j. And $R(e_{ik}, e_{ij})$ will be added to M'. If neither of e_{ik} and e_{ij} has matching element, they will both be added to M' as well as the relationship $R(e_{ik}, e_{ij})$ between them in the last step of model merging.
4. ***Similarity preservation.*** Elements that are declared to be similar (but not equal) to one another in mapping from one model to another retain their separate identity in target model and are related to each other by some relationship. Formally, for each pair of similar but not equal elements $e_i \in M_i$ and $e_j \in M_j$. There exist elements e'_i and $e'_j \in M'$ and $M(e_i, e'_i)$, $M(e_j, e'_j)$. As $Eq(e_i, e_j)$ is not true, e_i and e_j correspond to different elements in M', so $e'_i \neq e'_j$. Suppose $R(e_i, e_j)$ is the relationship between e_i and e_j. As relationship preservation is proved, there must be a relationship $R(e'_i, e'_j)$ between e'_i and e'_j in M'.
5. ***Meta-meta-model constraint satisfaction.*** There are meta-meta-model conflicts caused by one-type constraint and no-cycle constraint in model merging, we solve these problems by appointing a prior model and picking the elements in the prior model when conflicts occur. As the prior model is a correct UML model that satisfies all meta-meta-model constraints, the merged model M' satisfies them too.
6. ***Extraneous item prohibition.*** Other than the elements and relationships specified in source models, no additional elements or relationships exist in merged model. Formally, for each model element $e' \in M'$, $e' = e_i \in M_i$ if e_i has no matching element or $e' = MME_i$ which is a merged element whose properties and methods all come from

elements in $MES_i \in \{M_1, M_2, \ldots, M_n\}$. For each relationship $R(e'_i, e'_j) \in M'$, there exist $e_i, e_j \in M_i$, $M(e_i, e'_i)$, $M(e_j, e'_j)$ such that $R(e'_i, e'_j) = R(e_i, e_j) \in M_i$.

7. ***Property preservation.*** For each element property would be presented only if property of element is mapped exactly to element in target model. The goal is to prove for each element $e' \in M'$, e' has property p if and only if $\exists e_i \in M_i$, $M(e_i, e')$ and e_i has property p. Suppose that $\exists e_i \in M_i$, $M(e_i, e')$ and e_i has property p. If e' is not a merged element, then $e' = e_i$, so if e has property p, e' has p too. If e' is a merged element in matching element ME_i, and e_i is the prior element PE_i, then e' has all properties of e_i. If e_i is not PE_i, $\exists PE_i \in M_j$, and PE_i has the property p' which is the same property in different representation with p, such that e' has p' instead of p. If there is no such p' , then e' has p. If for each $e_i \in M_i (1 \leq i \leq n)$, $M(e_i, e')$, e_i does not have property p, then e' does not have property p.
8. ***Value preference.*** For each element in merged model M', its property value is chosen from mapping elements. Suppose that $\exists e_i \in M_i$, $M(e_i, e')$ and e_i has property p. For each element $e' \in M'$, if e' is not a merged element, then $e' = e_i$, $V(p, e_i) = V(p, e')$. If e' is a merged element in matching element ME_i, and e_i is the prior element PE_i, then $V(p, e_i) = V(p, e')$. If e_i is not PE_i, $\exists PE_i \in M_j$ has the property p' which is the same property in different representation with p, then $V(p', PE_i) = V(p', e')$. So for each $e' \in M'$, $p(e')$ is chosen from the mapping element corresponding to e' or the prior element PE_i corresponding to e'.

7. Case study

We implement a prototype tool of the proposed approach in Java. The source code of the tool is available online [54]. The code dictionary and the view of the proposed N-way model merging tool are given in Figure 8. This tool can be used to merge different versions of UML class diagrams modeled in the famous Papyrus modeling environment [42]. Users are supposed to run this tool in the Eclipse IDE. First, users need to move the different versions of UML class diagrams into the folder named $Models$ in the directory of the tool. Then, users need to set the relations of the UML versions as the example given in the file $version\_r$ in the $Models$ folder. After refreshing the tool view in Eclipse, users can see the diagram of the versions and the list of the versions as shown in Figure 8. Users need to select the versions to be merged by selecting corresponding checkbox. By editing the priority file following the example in the $Models$ folder, priority can be set to facilitate conflict prevention based on priority. After that, users need to click the button $Merge\ Selected\ Versions$ to accomplish model merging.

The proposed ABCMatch algorithm is a search-based meta-heuristic algorithm to search for the optimal matching solution. So we evaluate the algorithm by comparing it with existing novel meta-heuristic algorithms. The implemented tool merge the models according to the optimal matching solution. We evaluate the scalability of the tool by inviting participants to use the tool.

The purpose of the case study is to evaluate the effectiveness of the proposed method in real-world collaborative modeling. However, existing N-way model merging methods [6, 27, 28, 40] cannot directly process and generate complete UML diagrams as our tool does, they require a lot of manual processing. In addition, these methods do not provide publically available tools for us to compare with. EMFStore [4] is one of the most widely-used

version control systems in real-world collaborative modeling and it supports two-way model merging, so we choose it to compare our tool with.

This section evaluates the proposed approach by answering the following research questions.

- **RQ1.** Compared with the existing version control system, is the proposed N-way model merging approach more effective?
- **RQ2.** What are users' views on the usefulness of the proposed N-way model merging approach compared with the existing version control system?
- **RQ3.** How is the performance of the proposed ABCMatch algorithm compared with existing novel meta-heuristic algorithms?

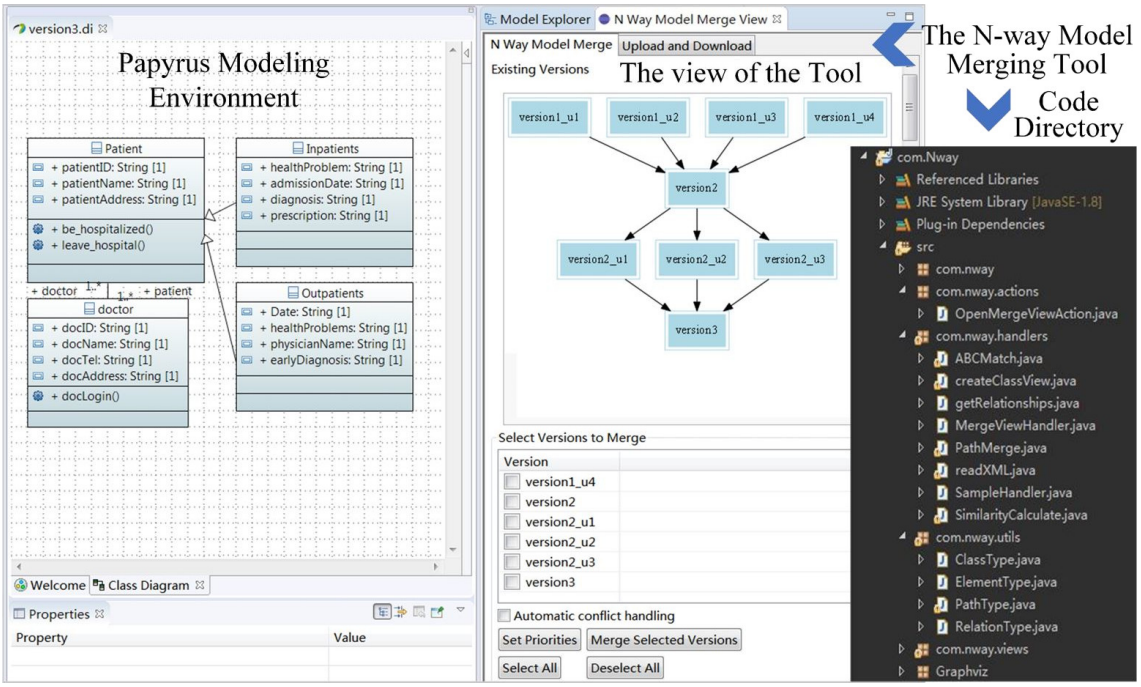


Figure 8. The code dictionary and the view of the proposed N-way model merging tool

7.1. Experiment

7.1.1. Participants

We selected 30 participants from the College of Computer Science and Technology of Nanjing University of Aeronautics and Astronautics. The selected participants are graduate students majoring in software engineering, The participants meet the following requirements:

1. At least 2 years of software modeling experience.
2. Have experience in using the model management tool EMFStore [4].

7.1.2. Modeling tasks

We designed the following two tasks with different workloads:

Task 1. Use the UML class diagram to model the following information in a library information system: (1) system users such as administrators, teachers, and students. (2)

system functions available to each type of user. Require a minimum of 40 classes in the submitted model.

Task 2. Use the UML class diagram to model the following information in a common hospital information system: (1) departments, (2) medical staff, (3) system functions available to medical staff, (4) wards, (5) patients, (6) system functions available to patients. Require a minimum of 80 classes in the submitted model.

We divided the participants into two groups, Group 1 and Group 2. Based on the information provided by the mentors of the participating students, we tried to make the modeling levels of the two groups as equal as possible. Group 1 and Group 2 were the experiment group and the control group, respectively, and were supposed to work on the same tasks with the N-way model merging tool developed in this paper and EMFStore [4], respectively. We asked participants to record the following information during the experiment: (1) the time taken to complete the task, (2) the number of model merges performed, (3) the number of conflicts, and (4) the total time taken for resolving conflicts, including the time taken for group discussion, the time taken for modifying conflicting models and the time taken for remerging models using the tool. In addition, we asked participants in the same group to work together in the same period of time each day for modeling, model merging, and group discussion to facilitate time consumption statistics.

7.1.3. Questionnaire

To compare participants' views of the tools, we designed the following questionnaire:

1. How useful is the model merging function supported by the tool in collaborative modeling?
   A. Very useful; **B.** Useful; **C.** A little useful; **D.** Not useful.
2. How useful is the conflict handling function supported by the tool in collaborative modeling?
   A. Very useful; **B.** Useful; **C.** A little useful; **D.** Not useful.
3. Which tool do you prefer as the model merging tool for collaborative modeling?
   A. The N-way model merging tool; **B.** EMFStore.

7.1.4. Evaluation of ABCMatch

In recent years, numerous novel meta-heuristic algorithms have been proposed to solve optimization problems in various fields. However, most of these algorithms are not suitable for solving the N-way model matching problem. This is because in the N-way model matching problem, each matching solution consists of multiple matching paths, and each matching path contains a set of UML classes. When using meta-heuristic algorithms to solve the N-way model matching problem, the locations of different agents are supposed to represent different matching solutions. However, most of the existing novel meta-heuristic algorithms update the locations of agents through complex numerical calculations, which cannot represent the update of the matching paths and the UML classes contained in matching paths in each model matching solution. Therefore, this paper cannot compare ABCMatch with the latest meta-heuristic algorithms which use complex numerical calculations to update locations of agents, such as Sparrow Search Algorithm (SSA) [21], Honey Badger Aptimization (HBA) [23], and Northern Goshawk Optimization (NGO) [24].

As a meta-heuristic algorithm, the reason why the ABC algorithm can be improved to solve the N-way model matching problem is that ABC updates the locations of agents by

searching adjacent locations with higher fitness without complex numerical calculations. Therefore, this paper can improve ABC and propose ABCMatch to solve the N-way model matching problem. ABCMatch regards two matching solutions with only one matching path to be different as neighbors when updating the locations of agents. We implemented the ABCMatch algorithm in Eclipse using Java. The parameter configuration of the algorithm is shown in Table 5.

Table 5. Parameter configuration

Algorithms	Description	Parameters	Value
GA[15]	Population size	PG	90
	Crossover rate	PC	0.9
	Mutation rate	PM	0.01
EHO[20]	Population size	PE	90
	Number of clans in elephant population	NClan	9
	Number of elephants in each clan	Nc	10
	Random number in the separating process	R	[0, 1]
ABCMatch	Population size	PA	90
	Number of employed bees	Ne	30
	Number of onlookers	No	30
	Number of scouts	Ns	30
	Initial number of matching paths in each matching solution	S	10

In order to evaluate the performance of the proposed ABCMatch algorithm by comparing it with existing novel meta-heuristic algorithms in the literature, we analyzed 23 existing state-of-the-art meta-heuristic algorithms to find the algorithms that can be used to solve the N-way model matching problem. These 23 algorithms are Genetic Algorithm (GA) [15], Gray Wolf Optimization (GWO) [16], Symbiotic Organisms Search (SOS) [17], Whale Optimization Algorithm (WOA) [18], Farmland Fertility Algorithm (FFA) [19],Elephant Herding Optimization (EHO) [20], Sparrow Search Algorithm (SSA) [21], Tunicate Swarm Algorithm (TSA) [22], Honey Badger Optimization (HBA) [23], and Northern Goshawk Optimization (NGO) [24] Ant Colony Optimization (ACO) [55], Particle Swarm Optimization (PSO) [56], Invasive Weed Optimization (IWO) [57], Firefly Algorithm (FA) [58], Fruit Fly Optimization (FFO) [59], Flower Pollination Algorithm (FPA) [60], Moth-Flame Optimization (MFO) [61], Crow Search Algorithm (CSA) [62], Dragonfly algorithm (DA) [63], Grasshopper Optimization Algorithm (GOA) [64], Spotted Hyena Optimization (SHO) [65], Emperor Penguin Optimization (EPO) [66], Butterfly Optimization Algorithm (BOA) [67]. Among the above algorithms, only GA[15] and EHO[20] do not need to update the positions of agents through complex numerical calculations, so they can be used to solve the N-way model matching problem.

Genetic algorithm (GA) [15]: GA is a classic meta-heuristic algorithm inspired by natural evolution theory. GA simulates the phenomena of replication, crossover and mutation in natural selection and genetics. Starting from any initial population, through random selection, crossover and mutation operations, it generates a group of individuals more suitable for the environment, so that the population evolves to a region with higher adaptability in the search space, and continues to reproduce and evolve from generation to generation, and finally converges to a group of individuals most suitable for the environment, so as to obtain the optimal solution of the problem. To compare ABCMatch with GA, we implemented GA using Java in Eclipse, and searched for the optimal solution of the N-way

model matching problem by selecting, crossing and mutating the matching paths in the model matching solutions. The parameter configuration of GA implemented in this paper is shown in Table 5.

Elephant Herding Optimization (EHO) [20]: EHO is an advanced swarm intelligence optimization algorithm that has been applied to optimization problems in many fields. This algorithm mainly simulates the herding behavior of elephant groups. In nature, an elephant group can be divided into multiple clans, and each clan has a matriarch as the leader. Elephants belonging to different clans live under the leadership of the matriarch (the best position in the clan). EHO has two operations: clan updating and separating. In the clan updating operation, the position of each elephant is updated according to its position and the position of the matriarch. In the separating operation, elephants with the worst finesses will be moved to new locations to increase the global search ability of the population. To compare ABCMatch with EHO, we implemented EHO in Eclipse using Java. In the implemented EHO algorithm, we regarded multiple sets of model matching solutions as multiple clans in an elephant group and took the best matching solution in each set as the matriarch in each group. For separating, new matching paths were randomly generated to replace the matching solutions with poor matching degrees. The parameter configuration of the EHO algorithm implemented in this paper is shown in Table 5.

Table 6. Details of the data sets

Data sets		Number of class diagrams	Number of matching paths	Number of class diagrams that can be matched
Class diagrams of the library information system (Task 1)	ModelSet1	332	25	233
	ModelSet2	380	28	271
	ModelSet3	417	32	300
	ModelSet4	422	32	304
	ModelSet5	426	33	308
Class diagrams of the hospital information system (Task 2)	ModelSet6	701	60	589
	ModelSet7	785	69	651
	ModelSet8	803	73	702
	ModelSet9	812	74	710
	ModelSet10	820	76	718

Data sets. In order to evaluate the performance of ABCMatch in solving the N-way model matching problem, this paper uses the class diagrams of the library information system and the hospital information system established by the participants in Task 1 and Task 2 as the experimental data sets, and compared the ABCMatch algorithm proposed in this paper with Genetic algorithm (GA) [15] and Elephant Herding Optimization (EHO) [20]. The details of the data sets are shown in Table 6. In Task 1, Group 1 conducted a total number of 13 times of model merging, and we randomly selected five times among them to collect the models to be merged. The selected models are from the first, the third, the tenth, the eleventh, and the thirteenth times of model merging and we named them as ModelSet1 to ModelSet5 in sequence as the experimental data sets. In Task 2, Group 1 conducted a total number of 29 times of model merging. We randomly selected the models from the fourth, the tenth, the thirteenth, the twenty-second and the twenty-ninth times of model merging, and named them as ModelSet6 to ModelSet10 in sequence as the experimental data sets.

Table 6 presents the details of each model set, including the total number of class diagrams in the model set, the number of matching paths in the model set, and the total number of class diagrams that can be matched in the model set. In order to avoid the statistics bias, the results of each algorithm are averaged over 30 runs.

Experimental environment. The experiment was performed on 64-bit Windows desktop computer with the following configuration: 3.19 GHz CPU and 16 GB RAM.

7.2. Results and discussion

RQ1.Compared with existing version control system, is the proposed N-way model merging approach more effective?

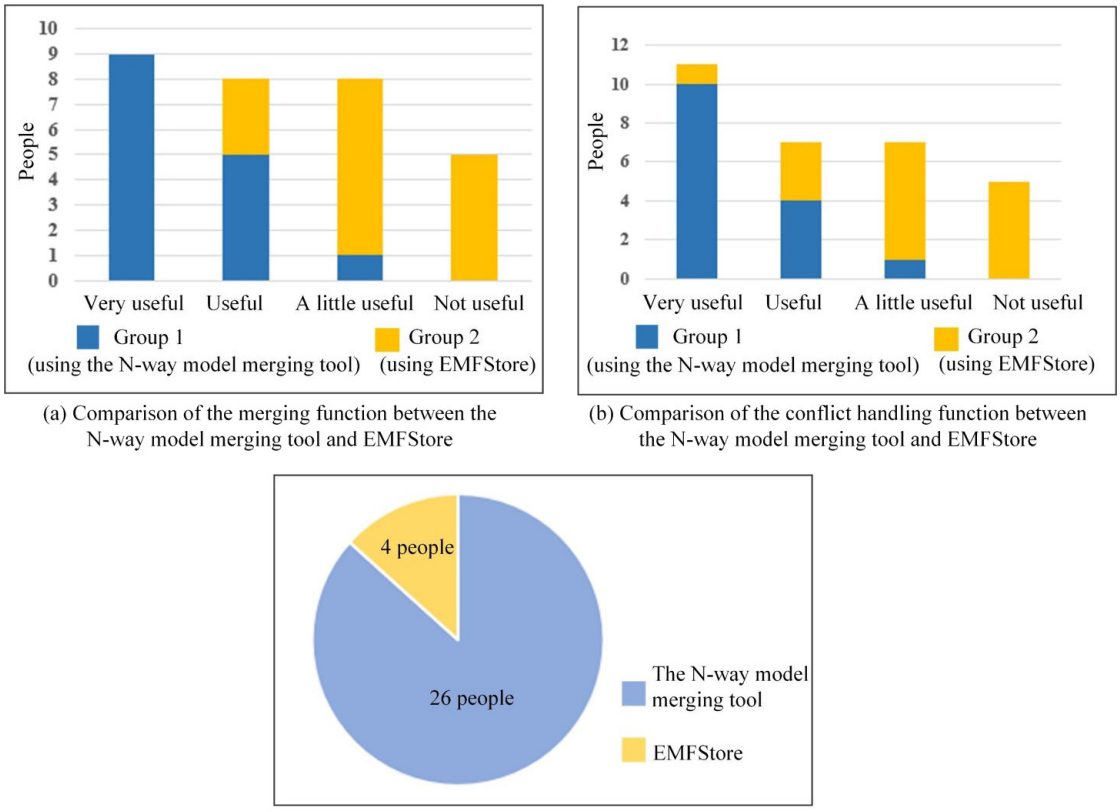
Table 7. Experimental data statistics of the two groups

		Group 1	Group 2
Task 1	Total Time Taken	12 h (6 days, 2 hours per day)	14 h (7 days, 2 hours per day)
	Total Number of Model Merging	13	131
	Total Number of Conflicts	92	96
	Total Time Taken for Resovling Conflicts	Total: 3 h 16 min Group Discussion: 1 h 50 min Modeling: 1 h 10 min Merging by Tool: 16 min	Total: 5 h 13 min Group Discussion: 2 h 10 min Modeling: 1 h 15 min Merging by Tool: 1 h 48 min
		Group 1	Group 2
Task 2	Total Time Taken	24 hours (12 days, 2 hours per day)	30 hours (15 days, 2 hours per day)
	Total Number of Model Merging	29	307
	Total Number of Conflicts	284	302
	Total Time Taken for Resovling Conflicts	Total: 9 h 22 min Group Discussion: 5 h 15 min Modeling: 3 h 10 min Merging by Tool: 57 min	Total: 15 h 51 min Group Discussion: 7 h 20 min Modeling: 4 h 18 min Merging by Tool: 4 h 13 min

Two groups of participants completed their tasks and submitted the models and related experimental records. The models built by Group 1 have 43 classes (Task 1) and 82 classes (Task 2), respectively, and the models built by Group 2 have 44 classes (Task 1) and 83 classes (Task 2), respectively. As we expected, the complexity of models built by the two groups is relatively similar due to our design when grouping participants. The statistical results of the experiment are given in Table 7. The results show that Group 1 (using the N-way model merging tool) spent less time on both tasks than Group 2 (using EMFStore [4]). And the time consumption gap is greater in the second task. The reason is that Group 2 spent more time on merging models using EMFStore when resolving conflicts, taking 1 hour and 48 minutes in Task 1 and 4 hours and 13 minutes in Task 2. This is much more than that of Group 1 which spent only 16 minutes (Task 1) and 57 minutes (Task 2) using the proposed N-way model merging tool.

The essential reason why our tool is more effective than EMFStore [4] is that EMFStore cannot support N-way model merging. Each group member needs to submit the model one by one. When conflicts occur, each member needs to wait for others to deal with the conflicts immediately, and only after resolving the conflicts can the next merge be carried out. Unlike EMFStore, our tool supports N-way model merging thus saving the extra

waiting time of submitting the model one by one. The results show the applicability of our approach in merging real-world collaborative models.



(a) Comparison of the merging function between the N-way model merging tool and EMFStore

(b) Comparison of the conflict handling function between the N-way model merging tool and EMFStore

(c) Comparison of user favoritism between the N-way model merging tool and EMFStore

Figure 9. Statistics on the results of the answers to the questions in the questionnaire

RQ2. What are users' views on the usefulness of the proposed N-way model merging approach compared with the existing version control system?

According to the statistical results of questionnaires given in Figure 9, most participants think that the N-way model merging tool proposed in this paper is more useful than the existing version control system EMFStore [4] in model merging and conflict handling and are more willing to choose our tool as the model merging tool.

As shown in Figure 9a, most of the participants (14 out of 15) in Group 1 (using the N-way model merging tool) stated that the N-way model merging tool was very useful or useful. And only one participant stated that the N-way model merging tool is a little useful. Compared with the N-way model merging tool, most participants in Group 2 (using EMFStore) found that the model merging function provided by EMFStore was not as useful as they had expected. And only a few participants (3 out of 15) stated that EMFStore's model merging function was useful. The results show that most participants agreed that the N-way model merging tool is useful and EMFStore is not useful enough. This is because a large number of models were required to be merged in the experiment. Unlike the N-way model merging tool which can merge a large number of models at one time, EMFStore requires more manual efforts to merge these models.

As shown in Figure 9b, most of the participants (14 out of 15) using the N-way model merging tool stated that the conflict handling function provided by the N-way model

merging tool was very useful or useful. In contrast, only a few participants (4 out of 15) felt that the conflict handling function of EMFStore was useful or very useful. This is because unlike EMFStore, the N-way model merging tool supports to show all the conflicts between the models to be merged at one time, which greatly improves the efficiency of conflict handling.

Figure 9c shows the comparison results of participants' favoritism for the N-way model merging tool and EMFStore. Most of the participants (26 of 30) prefer to use the N-way model merging tool in collaborative modeling. They stated that the N-way model merging tool can improve the efficiency of model merging when there are a large number of models to be merged. Other participants (4 out of 30) chose EMFStore because they were used to using EMFStore to manage their models.

RQ3. How is the performance of the proposed ABCMatch algorithm compared with existing novel meta-heuristic algorithms?

Table 8 shows the model matching accuracy and the time cost of ABCMatch, GA [15], and EHO [20] when the maximum model matching degree tends to be stable. The calculation method of model matching accuracy is shown in Equation (9). Where $N_{\text{correct\_path}}$ is the number of the correct matching paths in a matching solution, and $N_{\text{total\_path}}$ represents the total number of paths in a matching solution. A matching path is a correct matching path if and only if all the UML classes in this matching path are different versions of the same UML class.

$$Accuracy = \frac{N_{\text{correct\_path}}}{N_{\text{total\_path}}} \times 100\% \tag{9}$$

Table 8. Comparison of the model matching results between ABCMatch, GA [15], and EHO [20]

Data sets	ABCMatch		GA[15]		EHO[20]	
	Accuracy [%]	Time [min]	Accuracy [%]	Time [min]	Accuracy [%]	Time(min)
ModelSet1	98.0341	0.9687	95.6442	1.4283	96.5336	1.2557
ModelSet2	96.8774	0.9841	95.3084	1.4392	95.4013	1.2892
ModelSet3	97.8759	1.0259	94.2405	1.5870	96.7265	1.3066
ModelSet4	98.1253	1.0373	94.8720	1.5932	96.5449	1.3592
ModelSet5	96.9697	1.0429	93.3201	1.6079	95.1102	1.3623
ModelSet6	97.2340	1.8214	95.1853	3.1047	94.7228	2.9968
ModelSet7	98.0599	1.9028	94.9275	3.2824	96.0849	3.0294
ModelSet8	97.0538	2.0215	96.0284	3.2879	95.3972	3.0851
ModelSet9	98.1667	2.0250	95.0828	3.2901	95.9601	3.0883
ModelSet10	97.8613	2.0316	93.9238	3.2962	94.9730	3.0935

As shown in Table 8, for the five model sets ModelSet1–ModelSet5 from Task 1, the average value of the maximum model matching accuracy that GA and EHO can achieve when they tend to be stable are 94.6770% and 96.0633%, respectively. Compared with GA and EHO, the ABCMatch algorithm proposed in this paper can achieve a higher model matching accuracy of 97.5765% in a shorter time. For the five model sets ModelSet6-ModelSet10 from Task 2, the average value of the maximum accuracy that GA and EHO can achieve when they tend to be stable are 95.0296% and 95.4276%, respectively. Compared with GA and EHO, ABCMatch can achieve a higher model matching accuracy of 97.6751% in a shorter time. For all data sets in the experiment, ABCMatch has better performance than GA and EHO. The average accuracy of model matching has increased by 2.7725% and 1.8804% compared with GA and EHO, respectively. And the average time cost

has decreased by 0.9056 mins and 0.7005 mins compared with GA and EHO, respectively. The reason why ABCMatch performs better than EHO and GA is that ABCMatch can maintain a good balance between global search and local search through the cooperation of the employed bees, onlookers, and scouts. Unlike ABCMatch, GA has poor local search ability due to its randomness, so its performance is not as good as ABCMatch. Although EHO has a strong local search ability, its performance is not as good as the ABCMactch algorithm because of its poor global search ability. This problem makes it easy to fall into local optimum.

7.3. Threats to validity

In this section, we analyze the threats to the validity of the case study using the method [68] proposed by Wohlin et al.

1. ***Internal validity.*** The internal threat is that the modeling level of the participants could affect the results of the experiment. We set the modeling level that the participants need to achieve and tried to make the modeling levels of the two groups as equal as possible to relieve this threat.
2. ***External validity.*** The external validity means that the conclusion drawn from existing examples in the experimental data set is also valid for out-of-set examples [68]. Our experiments assume that the collaborative modeling is performed by a large group of people in the same period of time together. In this case, our approach is more effective than existing version control tools. We acknowledge that existing version control tools are more effective in the case that there is no need for a large group of users to resolve conflicts and merge models together during the same period of time.

The limited sample size poses a threat to the validity of our experimental results. Due to the limited time and manpower, in the current research work, only two group of participants were involved in the experiment. In the future, we will invite external practitioners and researchers to use the proposed method and continuously improve it.

To mitigate the threats to external validity brought by the particular tasks delivered in the experiment, we had selected two common and representative models of different scales in different fields. The selected models are similar to other common information systems in the real world.

3. ***Construct validity.*** The construct validity reflects the degree to which the case study truly represents what needs to be studied according to the research questions [68]. The two research questions of this paper are the effectiveness and users' views on the usefulness of the proposed approach compared with the existing version control tool. For the first question, we let two different groups (an experiment group and a control group) work on the same tasks with the two different approaches. For the second question, we obtained users' views on the usefulness of the proposed approach by questionnaire. So this case study totally represents what needs to be studied according to the two research questions.
4. ***Conclusion validity.*** As the conclusions of case studies are usually drawn from several cases, the results suffer from the threat to conclusion validity [68]. In this case study, we target to simulate the real-world collaborative modeling process to demonstrate the effectiveness of the proposed approach compared with the existing version control system. However, this requires much manual modeling effort and time consumption. So it is difficult to provide a large number of examples.

8. Conclusion and future works

In this paper, we propose a novel N-way model merging approach and accompanying prototype tool for collaborative modeling. Our approach takes a set of model variants as inputs and generates a single global merged model by model comparison, model matching and model combination. For model comparison, this paper proposes a new calculation method for calculating the similarity of a group of model elements. For the most challenging step, matching, which is an NP-hard problem, we propose a novel N-way matching algorithm ABCMatch. Model combination is implemented by building a matching model which is then transformed into the merged model. Unlike other approaches, we reshuffle elements from distinct chains by extracting the prior element link and store it in matching models rather than breaking the chain into pieces. Theoretical analysis is given to prove that our approach satisfies the Generic Merge Requirements. A case study is conducted and the experiment results corroborate the effectiveness of the proposed approach. Compared with existing novel meta-heuristic algorithms GA and EHO which can be used to solve the N-way model matching problem, the proposed ABCMatch algorithm can obtain more accurate model matching solutions in a shorter time. The average model matching accuracy of ABCMatch is 2.7725% higher than GA and 1.8804% higher than EHO.

At present, the N-way model merging method proposed in this paper is only applicable to merging UML class diagrams and cannot be used to merge other types of models, such as UML sequence diagrams. This is because the model comparison method, the model matching method and the model combination method used in this paper are specially designed for UML class diagrams. In the future, we will extend the model merging method proposed in this paper to support the merging of other types of UML models.

References

[1] T. Stahl, M. Völter, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development – Technology, engineering, management*. Hoboken, NJ, USA: John Wiley and Sons, Inc., 2006.

[2] M. Franzago, D.D. Ruscio, I. Malavolta, and H. Muccini, "Collaborative model-driven software engineering: A classification framework and a research map," *IEEE Transactions on Software Engineering*, 2017, p. 1.

[3] J.E. Rumbaugh, I. Jacobson, and G. Booch, *The unified modeling language reference manual*. Reading, MA, USA: Addison Wesley Longman, Inc., 1999.

[4] M. Koegel and J. Helming, "EMFStore: a model repository for EMF models," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Vol. 2, 2010, pp. 307–308.

[5] E. Stepper, "CDO model repository," 2010.

[6] J. Rubin and M. Chechik, "n-way model merging," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. Association for Computing Machinery, 2013, pp. 301–311.

[7] F.S. Gharehchopogh, "Advances in tree seed algorithm: A comprehensive survey," *Archives of Computational Methods in Engineering*, Vol. 29, 2022, pp. 3281–3304.

[8] H. Mohammadzadeh and F.S. Gharehchopogh, "A multi-agent system based for solving high-dimensional optimization problems: A case study on email spam detection," *International Journal of Communication Systems*, Vol. 34, No. 3, 2021, p. e4670.

[9] F.S. Gharehchopogh, I. Maleki, and Z.A. Dizaji, "Chaotic vortex search algorithm: Metaheuristic algorithm for feature selection," *Evolutionary Intelligence*, Vol. 15, No. 3, 2022, pp. 1777–1808.

[10] T.S. Naseri and F.S. Gharehchopogh, "A feature selection based on the farmland fertility algorithm for improved intrusion detection systems," *Journal of Network and Systems Management*, Vol. 30, No. 3, 2022, p. 40.
[11] H. Mohammadzadeh and F.S. Gharehchopogh, "Feature selection with binary symbiotic organisms search algorithm for email spam detection," *International Journal of Information Technology and Decision Making*, Vol. 20, No. 01, 2021, pp. 469–515.
[12] F.S. Gharehchopogh, "Quantum-inspired metaheuristic algorithms: Comprehensive survey and classification," *Artificial Intelligence Review*, 2022, pp. 1–65.
[13] S. Ghafori and F.S. Gharehchopogh, "Advances in spotted hyena optimizer: A comprehensive survey," *Archives of Computational Methods in Engineering*, 2021, pp. 1–22.
[14] D. Karaboga, "Artificial bee colony algorithm," *Scholarpedia*, Vol. 5, No. 3, 2010, p. 6915.
[15] S. Katoch, S.S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, Vol. 80, 2021, pp. 8091–8126.
[16] S. Mirjalili, S.M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, Vol. 69, 2014, pp. 46–61.
[17] M.Y. Cheng and D. Prayogo, "Symbiotic organisms search: A new metaheuristic optimization algorithm," *Computers and Structures*, Vol. 139, 2014, pp. 98–112.
[18] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, Vol. 95, 2016, pp. 51–67.
[19] H. Shayanfar and F.S. Gharehchopogh, "Farmland fertility: A new metaheuristic algorithm for solving continuous optimization problems," *Applied Soft Computing*, Vol. 71, 2018, pp. 728–746.
[20] G.G. Wang, S. Deb, and L.d.S. Coelho, "Elephant herding optimization," in *3rd International Symposium on Computational and Business Intelligence (ISCBI)*. IEEE, 2015, pp. 1–5.
[21] J. Xue and B. Shen, "A novel swarm intelligence optimization approach: Sparrow search algorithm," *Systems Science and Control Engineering*, Vol. 8, No. 1, 2020, pp. 22–34.
[22] S. Kaur, L.K. Awasthi, A. Sangal, and G. Dhiman, "Tunicate swarm algorithm: A new bio-inspired based metaheuristic paradigm for global optimization," *Engineering Applications of Artificial Intelligence*, Vol. 90, 2020, p. 103541.
[23] F.A. Hashim, E.H. Houssein, K. Hussain, M.S. Mabrouk, and W. Al-Atabany, "Honey Badger Algorithm: New metaheuristic algorithm for solving optimization problems," *Mathematics and Computers in Simulation*, Vol. 192, 2022, pp. 84–110.
[24] M. Dehghani, Š. Hubálovskỳ, and P. Trojovskỳ, "Northern goshawk optimization: A new swarm-based algorithm for solving optimization problems," *IEEE Access*, Vol. 9, 2021, pp. 162 059–162 080.
[25] F.S. Gharehchopogh, M. Namazi, L. Ebrahimi, and B. Abdollahzadeh, "Advances in sparrow search algorithm: A comprehensive survey," *Archives of Computational Methods in Engineering*, Vol. 30, No. 1, 2023, pp. 427–455.
[26] Y. Huo, Y. Zhuang, J. Gu, S. Ni, and Y. Xue, "Discrete gbest-guided artificial bee colony algorithm for cloud service composition," *Applied Intelligence*, Vol. 42, 2015, pp. 661–678.
[27] W.K.G. Assunção, S.R. Vergilio, and R.E. Lopez-Herrejon, "Discovering software architectures with search-based merge of UML model variants," in *ICSR*, 2017.
[28] D. Reuling, M. Lochau, and U. Kelter, "From imprecise N-way model matching to precise N-way model merging." *The Journal of Object Technology*, Vol. 18, No. 2, 2019.
[29] M. Koegel, H. Naughton, J. Helming, and M. Herrmannsdoerfer, "Collaborative model merging," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. Association for Computing Machinery, 2010, pp. 27–34.
[30] H.K. Dam, A. Reder, and A. Egyed, "Inconsistency resolution in merging versions of architectural models," in *IEEE/IFIP Conference on Software Architecture*, 2014, pp. 153–162.
[31] P. Buneman, S. Davidson, and A. Kosky, "Theoretical aspects of schema merging," in *International Conference on Extending Database Technology: Advances in Database Technology*, 1992.
[32] R. Pottinger and P.A. Bernstein, "Merging models based on given correspondences," in *Proceedings VLDB Conference*, 2003, pp. 862–873.

[33] M. Sharbaf and B. Zamani, "Configurable three-way model merging," *Software: Practice and Experience*, Vol. 50, No. 8, 2020.
[34] C. Thao and E.V. Munson, "Using versioned trees, change detection and node identity for three-way XML merging," *Computer Science – Research and Development*, Vol. 34, No. 1, 2019, pp. 3–16.
[35] C. Debreceni, I. Rath, D. Varro, X. De Carlos, X. Mendialdua et al., "Automated model merge by design space exploration," in *Fundamental Approaches to Software Engineering*, 2016, pp. 104–121.
[36] F. Schwagerl, S. Uhrig, and B. Westfechtel, "Model-based tool support for consistent three-way merging of EMF models," *ACME*, 2013, p. 2.
[37] A. Schultheiß, P.M. Bittner, L. Grunske, T. Thüm, and T. Kehrer, "Scalable N-way model matching using multi-dimensional search trees," in *ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2021, pp. 1–12.
[38] M.S. Kasaei, M. Sharbaf, and B. Zamani, "Towards a formalism for specifying N-way model merging rules," in *27th International Computer Conference, Computer Society of Iran (CSICC)*, 2022, pp. 1–7.
[39] M. Boubakir and A. Chaoui, "A pairwise approach for model merging," in *Modelling and Implementation of Complex Systems*, S. Chikhi, A. Amine, A. Chaoui, M.K. Kholladi, and D.E. Saidouni, Eds. Cham: Springer International Publishing, 2016, pp. 327–340.
[40] Y. Jiang, S. Wang, K. Fu, W. Zhang, and H. Zhao, "A collaborative conceptual modeling tool based on stigmergy mechanism," in *Proceedings of the 8th Asia-Pacific Symposium on Internetware*, 2016, pp. 11–18.
[41] J. Martinez, T. Ziadi, T.F. Bissyande, J. Klein, and Y.L. Traon, "Automating the extraction of model-based software product lines from model variants," 2015, pp. 396–406.
[42] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard et al., "Papyrus UML: An open source toolset for MDA," in *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*. Citeseer, 2009, pp. 1–4.
[43] F.S. Gharehchopogh, M.H. Nadimi-Shahraki, S. Barshandeh, B. Abdollahzadeh, and H. Zamani, "CQFFA: A Chaotic Quasi-Oppositional Farmland Fertility Algorithm for Solving Engineering Optimization Problems," *Journal of Bionic Engineering*, Vol. 20, No. 1, 2023, pp. 158–183.
[44] F.S. Gharehchopogh, "An improved tunicate swarm algorithm with best-random mutation strategy for global optimization problems," *Journal of Bionic Engineering*, Vol. 19, No. 4, 2022, pp. 1177–1202.
[45] B. Abdollahzadeh and F.S. Gharehchopogh, "A multi-objective optimization algorithm for feature selection problems," *Engineering with Computers*, Vol. 38, No. Suppl 3, 2022, pp. 1845–1863.
[46] Şaban Öztürk, R. Ahmad, and N. Akhtar, "Variants of Artificial Bee Colony algorithm and its applications in medical image processing," *Applied Soft Computing*, Vol. 97, 2020, p. 106799.
[47] U. Mansoor, M. Kessentini, P. Langer, M. Wimmer, S. Bechikh et al., "MOMM: Multi-objective model merging," *Journal of Systems and Software*, Vol. 103, 2015, pp. 423–439.
[48] A. Anwar, A. Benelallam, M. Nassar, and B. Coulette, "A graphical specification of model composition with triple graph grammars," in *Model-Based Methodologies for Pervasive and Embedded Software*, Vol. 7706, 2012, pp. 1–18.
[49] A. Koshima and V. Englebert, "RuCORD: Rule-based composite operation recovering and detection to support cooperative edition of (meta)models," in *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2015, pp. 1–7.
[50] H. Chong, R. Zhang, and Z. Qin, "Composite-based conflict resolution in merging versions of UML models," in *17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2016, pp. 127–132.
[51] J. Rubin and M. Chechik, "Combining related products into product lines," in *Fundamental Approaches to Software Engineering*, 2012, pp. 285–300.
[52] P. Jaccard, "The distribution of the flora in the alpine zone. 1," *New Phytologist*, Vol. 11, No. 2, 1912, pp. 37–50.

[53] D. Karaboga and B. Gorkemli, "A combinatorial artificial bee colony algorithm for traveling salesman problem," in *International Symposium on Innovations in Intelligent Systems and Applications*, 2011, pp. 50–53.
[54] "*N* Way Model Merging Tool," https://github.com/YETONG1219/NWay, [accessed 7 Nov. 2023].
[55] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, Vol. 1, No. 4, 2006, pp. 28–39.
[56] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 – International Conference on Neural Networks*, Vol. 4. IEEE, 1995, pp. 1942–1948.
[57] S. Karimkashi and A.A. Kishk, "Invasive weed optimization and its features in electromagnetics," *IEEE Transactions on Antennas and Propagation*, Vol. 58, No. 4, 2010, pp. 1269–1278.
[58] X.S. Yang and X. He, "Firefly algorithm: Recent advances and applications," *International Journal of Swarm Intelligence*, Vol. 1, No. 1, 2013, pp. 36–50.
[59] B. Xing, W.J. Gao, B. Xing, and W.J. Gao, "Fruit fly optimization algorithm," *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms*, 2014, pp. 167–170.
[60] X.S. Yang, M. Karamanoglu, and X. He, "Flower pollination algorithm: A novel approach for multiobjective optimization," *Engineering Optimization*, Vol. 46, No. 9, 2014, pp. 1222–1237.
[61] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowledge-based Systems*, Vol. 89, 2015, pp. 228–249.
[62] A. Askarzadeh, "A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm," *Computers and Structures*, Vol. 169, 2016, pp. 1–12.
[63] S. Mirjalili, "Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Computing and Applications*, Vol. 27, 2016, pp. 1053–1073.
[64] S.Z. Mirjalili, S. Mirjalili, S. Saremi, H. Faris, and I. Aljarah, "Grasshopper optimization algorithm for multi-objective optimization problems," *Applied Intelligence*, Vol. 48, 2018, pp. 805–820.
[65] G. Dhiman and V. Kumar, "Multi-objective spotted hyena optimizer: A multi-objective optimization algorithm for engineering problems," *Knowledge-Based Systems*, Vol. 150, 2018, pp. 175–197.
[66] G. Dhiman and V. Kumar, "Emperor penguin optimizer: A bio-inspired algorithm for engineering problems," *Knowledge-Based Systems*, Vol. 159, 2018, pp. 20–50.
[67] S. Arora and S. Singh, "Butterfly optimization algorithm: A novel approach for global optimization," *Soft Computing*, Vol. 23, 2019, pp. 715–734.
[68] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell et al., *Experimentation in software engineering: An introduction*. Kluwer Academic Publishers, 2000.

e-Informatica Software Engineering Journal (EISEJ) is an international, fully open access (CC-BY 4.0 without any fees for both authors and readers), blind peer-reviewed computer science journal using a fast, continuous publishing model (papers are edited, assigned to volume, receive DOI & page numbers, and are published immediately after acceptance without waiting months in a queue to be assigned for a specific volume/issue) without paper length limit. Our aim is to focus on empirical software engineering, as well as data science in software engineering.

The journal is published by *Wrocław University of Science and Technology* under the auspices of the *Software Engineering Section of the Committee on Informatics of the Polish Academy of Sciences*.

Aims and Scope

The purpose of **e-Informatica Software Engineering Journal** is to publish original and significant results in all areas of software engineering research.

The scope of **e-Informatica Software Engineering Journal** includes methodologies, practices, architectures, technologies and tools used in processes along the software development lifecycle, but particular stress is laid on empirical evaluation using well-chosen statistical and data science methods.

e-Informatica Software Engineering Journal is published online and in hard copy form. The on-line version is from the beginning published as a gratis, no authorship fees, open-access journal, which means it is available at no charge to the public. The printed version of the journal is the primary (reference) one.

Topics of interest

- Software requirements engineering and modeling
- Software architectures and design
- Software components and reuse
- Software testing, analysis and verification
- Agile software development methodologies and practices
- Model driven development
- Software quality
- Software measurement and metrics
- Reverse engineering and software maintenance
- Empirical and experimental studies in software engineering (incl. replications)
- Evidence-based software engineering
- Systematic reviews and mapping studies (see SEGRESS guidelines)
- Statistical analyses and meta-analyses of experiments
- Robust statistical methods
- Reproducible research in software engineering
- Object-oriented software development
- Aspect-oriented software development
- Software tools, containers, frameworks and development environments
- Formal methods in software engineering.
- Internet software systems development
- Dependability of software systems
- Human-computer interaction
- AI and knowledge based software engineering
- Data science in software engineering
- Prediction models in software engineering
- Mining software repositories
- Search-based software engineering
- Multiobjective evolutionary algorithms
- Tools for software researchers or practitioners
- Project management
- Software products and process improvement and measurement programs
- Process maturity models

Funding acknowledgements: Authors are requested to identify who provided financial support for the conduct of the research and/or preparation of the article and to briefly describe the role of the sponsor(s), if any, in study design; in the collection, analysis and interpretation of data; in the writing of the paper. If the funding source(s) had no such involvement then this should be stated as well.

The submissions will be accepted for publication on the base of positive reviews done by international Editorial Board and external reviewers.

English is the only accepted publication language. To submit an article please enter our online paper submission site.

Subsequent issues of the journal will appear continuously according to the reviewed and accepted submissions.

The journal is indexed by ISI Web of Science, Scopus, DBLP, Google Scholar, DOAJ, Index Copernicus, etc.

Paper copies of selected issues of the journal are available from our Publisher (please contact oficwyd@pwr.wroc.pl for details).

http://www.e-informatyka.pl/

e-Informatica

ISSN 1897-7979