International Journal of Computer Integrated Manufacturing

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tcim20

(AIAM2019) Artificial Intelligence in Software Engineering and inverse: Review

Mohammad Shehab, Laith Abualigah, Muath Ibrahim Jarrah, Osama Ahmad Alomari & Mohammad Sh. Daoud

To cite this article: Mohammad Shehab, Laith Abualigah, Muath Ibrahim Jarrah, Osama Ahmad Alomari & Mohammad Sh. Daoud (2020) (AIAM2019) Artificial Intelligence in Software Engineering and inverse: Review, International Journal of Computer Integrated Manufacturing, 33:10-11, 1129-1144, DOI: 10.1080/0951192X.2020.1780320

To link to this article: https://doi.org/10.1080/0951192X.2020.1780320

Published online: 25 Jun 2020.

INTERNATIONAL JOURNAL OF COMPUTER INTEGRATED MANUFACTURING
2020, VOL. 33, NOS. 10–11, 1129–1144
https://doi.org/10.1080/0951192X.2020.1780320

Taylor & Francis
Taylor & Francis Group

ARTICLE

Artificial intelligence in software engineering and inverse: review

Mohammad Shehab[a], Laith Abualigah[b], Muath Ibrahim Jarrah[c], Osama Ahmad Alomari[d] and Mohammad Sh. Daoud[e]

[a]Computer Science, Artificial Intelligence Department, Aqaba University of Technology, Aqaba, Jordan; [b]Faculty of Computer Sciences and Informatics, Amman Arab University, Amman, Jordan; [c]Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Parit Raja, Malaysia; [d]Department of Computer Engineering, Istanbul Gelisim University, Istanbul, Turkey; [e]Faculty of Engineering, Al Ain University of Science and Technology, Abu Dhabi, UAE

ABSTRACT

Artificial Intelligence (AI) and Software Engineering are considered as significant fields to solve various problems. However, there are weaknesses in certain problem-solving in each field. Thus, this paper is a broad-based review of using artificial intelligence (AI) to improve software engineering (SE), and vice versa. As well as it intends to review the techniques developed in artificial intelligence from the standpoint of their application in software engineering. The aim of this review is highlighted in how the previous study benefited from incorporating the advantages of both fields. The researchers and practitioners on AI and SE belong to a wide range of audiences from the domains of optimization, engineering, data mining, clustering, etc., who will benefit from this study and areas for potential future research.

ARTICLE HISTORY
Received 1 August 2019
Accepted 30 May 2020

KEYWORDS
Artificial intelligence; software engineering; software development process; artificial intelligence techniques

1. Introduction

Artificial intelligence (AI) is the capability of the digital computer to perform tasks in ways that are smarter than human ability (Shahkarami et al. 2014). It has two types based on performance strength. (i) Narrow AI which handles the subgroups of a possible scenario. All these subgroups are used to build strong AI (Yampolskiy and Spellchecker 2016). For example, in chess, all rules are entered manually, and the machine begins using these rules depending on the situation. Meanwhile, (ii) strong AI is used in complex systems where human intervention is not required, and the machine can think and execute tasks singlehandedly (Stewart 2015).

Meanwhile, software engineering (SE) consists of two expressions. Software refers to programs that include instructions to supply the required functionality, and engineering refers to the operations of the design and construction to determine the cost of efficient solutions. Thus, the definition of SE is a systematic approach for the design, development, implementation, and maintenance of a software system (Winter, Forshaw, and Ferrario 2018).

This paper aims to provide unusual guidelines to work with AI systems that can be used in determining problems incorporated with SE methods. Another objective is to obtain the specific AI methods suitable for producing appropriate software improvement processes (Shankari and Thirumalaiselvi 2014).

This review is organized as follows: Section 2 and 3 illustrate an overview of the Artificial Intelligence and Software Engineering, respectively. The publication growth, benefits, and related works of Artificial Intelligence and Software Engineering are shown in Section 4. Section 5 presents the discussion. Finally, Section 6 draws some concluding remarks and outlines several future research lines of interest.

2. Artificial intelligence

The Dartmouth conference in 1956 was the birthplace of AI (Hamet and Tremblay 2017). Various definitions of AI exist but all of them include the same purpose. Winston and Prendergast (1984) mentioned that AI is "the study of the computations that make it possible to perceive, reason, and act". According to Kurzweil et al. (1990), AI is "the art of creating machines that perform functions that require intelligence when performed by people". AI differs from most of psychology because of its emphasis on computation and it differs

CONTACT Mohammad Shehab moh.shehab12@gmail.com Computer Science, Artificial Intelligence Department. Aqaba University of Technology, Aqaba 77110, Jordan

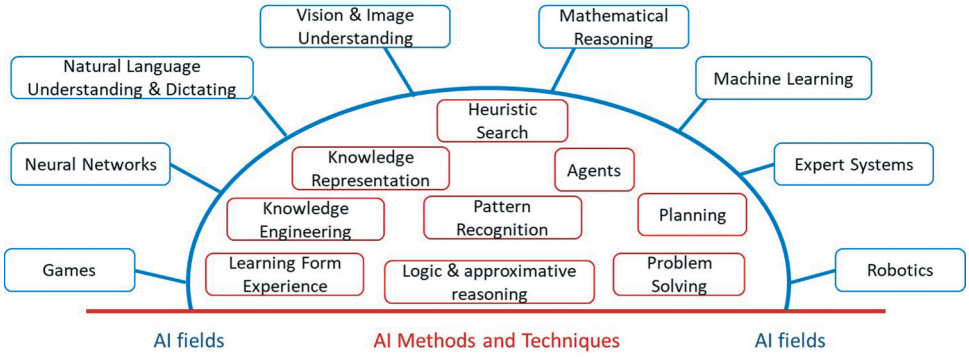

Figure 1. AI fields, methods, and techniques (Rech and Althoff 2004).

from most of computer science because of its emphasis on perception, reasoning, and action according to Abualigah et al. (2019).

AI applications are used to simulate human intelligence for providing help in problem solving and decision making. AI has been applied and is still utilized in various fields, such as economics, engineering, law, economics, medicine, and manufacturing (Shehab, Khader, and Al-Betar 2017). Figure 1 shows an overview of the fields, methods, and techniques of AI. AI has numerous advantages, such as making decisions with rapid thinking, replacing humans in certain jobs (such as dangerous tasks), and simplifying life (such as with the use of smartphones and GPS) (Chowdhury and Sadek 2012). However, AI also has flaws, such as being expensive, and it may cause adverse results (such as corruption or malfunction in robot armies and possibility of mass destruction) and create unemployment in certain sectors (Shehab et al. 2019).

3. Software engineering

SE is the application of engineering to the design, development, and maintenance of software. SE was initially introduced to address the issues of low-quality software projects. Problems arise when a software generally exceeds timelines, budgets, and reduced levels of quality. Thus, SE ensures that applications are constructed consistently, correctly, on time, on budget, and within requirements. The demand of SE also emerged to cater to the immense rate of change in user requirements and environment, on which applications should function well. Figure 2 shows the common keywords in the SE lifecycle.

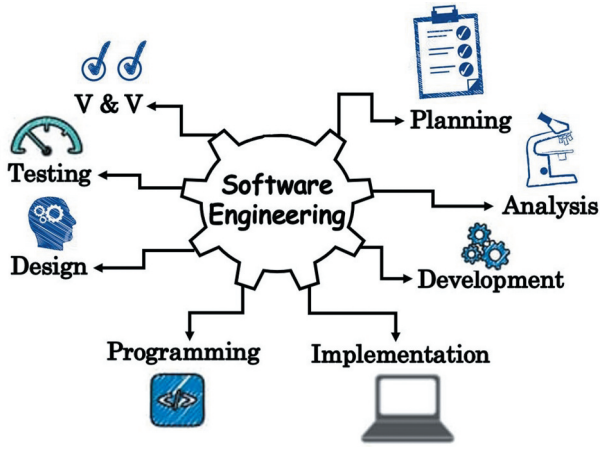

Figure 2. The keywords of the software engineering.

3.1. The software development process

This section shows the software engineering process followed by summarizing the common models in the software development life cycle.

3.1.1. Software process

A software process (also known as software methodology) is a set of related activities that leads to the production of the software (Eisty, Thiruvathukal, and Carver 2019). These activities may involve developing software from scratch or modifying an existing system. Any software process must include the following four activities:

(1) Software specification (requirements engineering): The major functionalities of the software and the constrains around them are defined.
(2) Software design and implementation: The software is designed and programmed.

(3) Software verification and validation: The software must conform to specifications and meet the customer needs.
(4) Software evolution (software maintenance): The software is modified to meet the customer needs and the changes in market requirements.

3.1.2. Software development life cycle (SDLC) model

A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes or for building empirically grounded prescriptive models (Rao, Kumar, and Reddy 2018). Table 1 presents the SDLC common models.

Table 1. Software development life cycle models.

Model	Advantages	Disadvantages	Ref
Waterfall	● Simple and easy to use and understand	● High amounts of risk and uncertainty	Kramer (2018), Shambour et al. (2018), Kuhrmann et al. (2017)
	● Easy to manage (i.e., each phase has specific deliverables)	● Poor for long and ongoing projects	
	● processed and completed one at a time	● Cannot accommodate changing requirements	
	● works well for smaller projects	● difficult to measure progress within stages	
Prototyping	● Missing functionality can be identified easily	● Insufficient analysis	Shambour, Abusnaina, and Alsalibi (2019), Falzone and Bernaschina (2018), Devadiga (2017)
	● Reduces time and cost	● User confusion	
	● Quicker user feedback is available	● Developer misunderstanding of user objectives	
	● Improved and increased user involvement	● Excessive Development Time	
Spiral	● Can changing requirements	● Risk of not meeting the schedule or budget	Shambour (2017), Boehm and Turner (2015), Krishnan (2015)
	● Users see the system early	● High cost	
	● Allows extensive use of prototypes	● Not useful for small projects	
	● helps in better risk management	● Not suitable for low risk projects	
Incremental	● Easy to test and debug	● Requires a good planning designing	Semeráth, Vörös, and Varró (2016), Varró et al. (2016), Abu-Hashem, Uliyan, and Abuarqoub (2017)
	● A customer can respond to each building	● Each iteration phase is inflexible and does not overlap each other	
	● Errors are easy to be identified	● Problems may arise pertaining to system architecture	
	● Thought the development stages changes can be done	● More expensive comparing to the waterfall model	
Iterative	● Can be developed parallel	● Required highly skilled resources for risk analysis	Meja Niño et al. (2018), Menghi, Rizzi, and Bernasconi (2018), Abu-Hashem et al. (2015)
	● Less costly to change the requirements	● End of project may not be known	
	● Easy of testing and debugging during smaller iteration	● Management complexity is more	
	● Easier to manage risk	● Not suitable for smaller projects	
Agile	● Very realistic approach to software development	● More risk of sustainability	Ringert et al. (2017), Shehab (2020a), Al-Zewairi et al. (2017)
	● Suitable for fixed or changing requirements	● Very high individual dependency	
	● Need minimal resource requirements	● Not suitable for handling complex dependencies	
	● Easy to manage	● Depends heavily on customer interaction	

3.2. Objectives of software engineering

The following points illustrate the main objectives of software engineering.

(1) *Reliability*: This attribute of software quality is the extent to which a program can be expected

to perform its desired function over an arbitrary time period (Qiang and Peña 2018).

(2) *Reusability*: A software product has good reusability if the different modules of the product can easily be reused to develop new products (Tahir et al. 2016).
(3) *Maintainability*: The software should be feasible to evolve to meet changing requirements (Jain, Sharma, and Ahuja 2018).
(4) *Testability*: Software establishes test criteria and evaluates software with respect to those criteria (Hassan et al. 2015).
(5) *Correctness*: A software product is correct if the different requirements as specified in the SRS document are correctly implemented (Chakraborty et al. 2018).
(6) *Adaptability*: Software allows differing system constraints and user needs to be satisfied by making changes to the software (Fink, Wyss, and Lichtenstein 2018).
(7) *Portability*: Software can be transferred from one computer system or environment to another (Bonati et al. 2015).

3.3. Challenges

SE utilizes a well-defined and systematic approach for software development. This approach is considered to be the most effective for producing high-quality software. However, despite this systematic approach in software development, some serious challenges are still faced by SE. Some of these challenges are listed below.

(1) The methods used to develop small- or medium-scale projects are inapplicable when developing large-scale or complex systems (Chapman 2018).
(2) Changes in software development are unavoidable. Currently, changes occur rapidly, and accommodating these changes to complete a software project is a major challenge faced by the software engineers (Malhotra and Bansal 2016).
(3) The advancement in computer and software technology has necessitated for the changes in the nature of software systems. Software systems that cannot accommodate changes are not considerably useful. Thus, a challenge in SE is to produce high-quality software that adapts to the changing needs within acceptable schedules. To meet this challenge, the object-oriented approach is preferred; however, accommodating changes to software and its maintenance within acceptable cost remains a challenge (Gui et al. 2015).
(4) Informal communications account a considerable portion of the time spent on software projects. Such wastage of time delays the completion of projects in the specified time (Johanson and Hasselbring 2018).
(5) The user generally has only a vague idea about the scope and requirements of the software system. This usually results in the development of software, which does not meet the user's requirements (Abad, Noaeen, and Ruhe 2016).
(6) Changes are usually incorporated in documents without following any standard procedure. Thus, verification of such changes often becomes difficult (Mistry 2017).
(7) The development of high-quality and reliable software requires the software to be thoroughly tested. Although thorough testing of software consumes the majority of resources, underestimating it deteriorates the software quality (Noureddine, Rouvoy, and Seinturier 2015).

In addition to the aforementioned key challenges, the responsibilities of system analysts, designers, and programmers are usually not well defined. Also, if the user requirements are not precisely defined, then software developers can misinterpret the meaning (Stark et al. 1999). All these challenges need to be addressed to ensure that the software is developed within the specified time and estimated costs and meets the requirements specified by the user.

4. Artificial intelligence and software engineering

This section illustrates the growth of the integration between AI and SE, the benefits of this integration, and the related works.

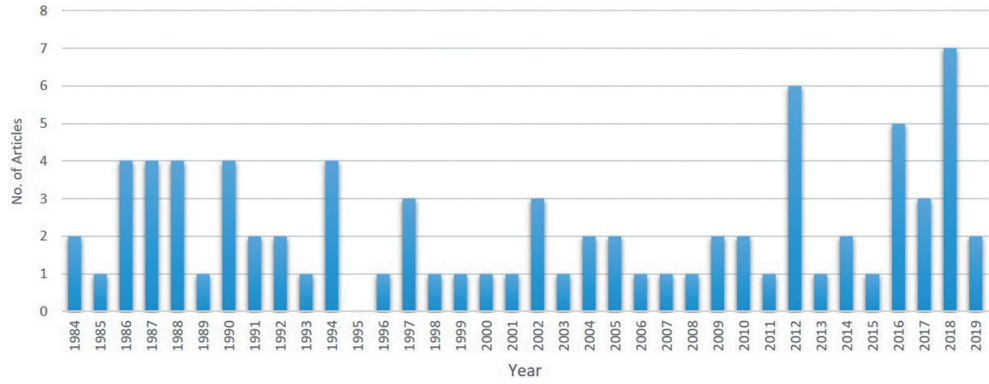

Figure 3. Publications of artificial intelligence and software engineering.

4.1. Growth of the integration between AI and SE

Several studies on AI and SE have been widely published. Figure 3 shows the number of published articles between 1984 and January 2019. The materials were collected using the keyword AI and SE. First, published articles were collected from highly reputed publishers, such as Springer, Elsevier, and IEEE, as well from other journals searched via Google Scholar. Second, the search results were classified per publishing date to show the growth of the integration between AI and SE.

4.2. Benefits

According to the father of AI, Marvin Minsky, "using SE is important to build strong AI systems". The creation of a huge AI system is expensive (Tenne and Goh 2010). Thus, problem analysis, modeling, implementation, test, and evaluation should be carefully performed. Utilizing AI has a positive effect on the performance of the SE field, such as rapid creation of minimum viable products, project management, automatic debugging, smart assistants, and automatic code generation and testing (Shehab et al. 2020).

SE is currently a global hotspot, and automation is the next trend. Software and engineers need to automate everything in technology. The role of the AI domain in the SE domain was validated in (Tamalika et al. 2017). Rather than humans manually generating all the software code to obtain services, users' need, and test automation, tools can help in generating the code and executing test and analysis for the code and can constantly develop as they obtain personal information. This paper reviews and summarizes the different benefits and drawbacks regarding automating SE. It also explains the position that factors can perform in a scheme.

The two sciences, that is, AI and SE, are the two major areas of the computer science domain. AI refers to creating intelligent machines, whereas SE is an information-intensive exercise that requires vast knowledge of the application area and the purpose of software. Shankari and Thirumalaiselvi (2014) proposed a survey for the procedures generated in AI from the attitude of their importance in SE.

4.3. Related works

This section introduces three categories of related works: (i) AI using SE, (ii) SE using AI, (iii) and integration of AI and SE (where the utilization ratio of each field is nearly equal). These categories were divided on the basis of the contribution of each previous study.

4.3.1. Artificial intelligence using software engineering

AI software is a fact in the computer science domain but only for specific classes of computer science problems. Usually, difficulties in AI are different from those of traditional SE. The variations proposed in (Partridge 1992) include various program improvement approaches for difficulties and problems in AI, that is, programs with the purposes of functional software (such as reliability, maintainability, robustness, and usability) are not produced rapidly. In addition, the difficulties and problems of machine learning techniques must be solved (to some level) before the full prospect of AI can be achieved, without expecting the resultant self-adaptive software to increase the difficulties in AI. The realization of the full prospect of AI in working software expects some essential discoveries in basic difficulties and problems

in AI and a suitable software improvement approach (Abualigah et al. 2020).

SE for game purposes is a class of important applications that are utilized for entertainment, amusement, and earnest purposes and used in various regions and domains, such as education, science, business, gaming, and healthcare. The game improvement process varies from the popular software improvement method because it includes interdisciplinary field actions. Therefore, SE methods are still necessary for game improvement progress because they can encourage developers to obtain maintainability, flexibility, usability, low effort and cost, readability, and reliable design. The aim of this comparison is to evaluate the well-known published studies on the game improvement SE method and determine sections that still require further investigation by researchers. In the investigation, a systematic literature analysis methodology was conducted in (Aleem, Capretz, and Ahmed 2016) using well-known digital publishers. An extensive number of studies have reported the creation phase of the life cycle of the game improvement SE method supported by the preproduction phase. By contrast, the postproduction aspect has been less researched than the preproduction and creation aspects. The outcomes of this study recommend that the game improvement SE method, especially the postproduction aspect, still requires further research.

Numerous difficulties and problems in SE include determining optimal solutions from a collection of candidate solutions. Such techniques usually need stakeholders, such as developers, programmers, and testers, to define decisions across various characteristics or purposes that are to be optimized (that is, searching and finding the optimal solution from the available solutions). However, in many instances, stakeholders should show such decisions in easy and qualitative terms. The survey conducted in (Santhanam 2016) reported that qualitative optimization methods can be beneficial to handle dilemmas within SE. Furthermore, a new optimization technique was developed to rely on stakeholders' qualitative decisions, thereby leveraging modern approaches in decision-theoretic AI techniques. This study proved the beneficial uses and produced conjunctions between qualitative judgment hypothesis and SE.

In the past few years, deep learning (DL) methods have gained enormous success and benefits. They have also reached excellent reputation in many different applications, such as intelligent machines, data mining, image processing, text mining, speech processing techniques, and therapeutic diagnostics. Deep neural networks are the important driving power behind the modern success of DL methods. However, black box testing still requires interpretability and further knowledge. This problem results in multiple open safety and protection issues with tremendous and essential requirements on accurate techniques and SE system for quality improvement. A large amount of studies have explained that state-of-the-art DL schemes yield weaknesses and vulnerabilities that can head to critical failure and difficulties, especially when used to real-world safety-critical attention. Large-scale studies have introduced an article repository of 223 related works in (Ma et al. 2018) regarding the quality assurance, safety, security, and understanding of DL. These studies, from a software quality assurance aspect, have determined difficulties and future possibilities toward safe DL in the SE domain. This work and the conducted paper repository covered the track for the SE researchers with respect to solving the important technical requirement for safe AI applications.

4.3.2. Software engineering using artificial intelligence

Tunio et al. (2018) proposed classical planning of AI techniques to promote quality and alleviate the burden of the crowdsourcing software development (CSD) platform and CSD developers by searching in the open tasks and optimizing the matching process, which matches the CSD developers to develop and provide a solution for the given task. The results demonstrated that the automatic planning has a remarkable influence and suitable efficiency of matching the task and personality than human work.

In SE automation, introducing a solution for SE problems in one piece is complex. Thus, C.V. Ramamoorthy and Shim (1991) presented AI techniques for SE research by using the principle of divide-and-conquer approach and providing the criteria for dividing SE problems. Moreover, they provided recommendations on how to conduct research in AI for SE and evaluate the results in accordance with scalability, generality, and combinability.

In software development phase, Ammar, Abdelmoez, and Hamdi (2012) applied fuzzy logic in

software testing to address the uncertainty experienced in this phase.

In (Claypool and Claypool 2005), the authors proposed a new approach that allows gaining further understanding and effective education of SE using game theories. In this study, the aforementioned objectives were fulfilled by using a computer game-based project. Particularly, project-based and set game-centric modules were combined in managing students to (1) engage actively in several phases of software life cycle; (2) allow them to track real issues experienced in the project and team management during the course that extended to a two-semester project; and simultaneously (3) expose them to various aspects of computer game design. The preliminary results proved the effectiveness of the proposed approach to improve the class participation and performance in SE course.

A group of researchers have devoted their efforts in creating auto-coding software called DeepCoders by using AI system (Zohair 2018). The DeepCoders can develop a working program after referring to massive database of codes. However, the current version of this software is limited to write a mini program, which does not exceed five lines of codes, but may be extended in the succeeding years as declared by the developers.

Many researchers have utilized AI-based approaches to address SE problems, such as requirement and design, maintenance, and testing (Zhang, Finkelstein, and Harman 2008; Räihä 2010; Shehab 2020b). The application of optimization search approaches in solving such problems is widely known as search-based SE (SBSE) and was proven to be successful and applicable (Harman 2012).

Participation in SE activities requires intense human effort. AI techniques have been widely used to automate these activities. Traditionally, the starting points in designing AI techniques rely on human's domain knowledge. Thereafter, findings are interpreted or verified by human users. However, another direction of research has focused on using user feedback to be incorporated with AI techniques to improve their performance further. Xie (2013) proposed cooperative testing and analysis involving on human-tool cooperation (embedding of human-assisted and human-centric computing) and human-human cooperation. Such mechanism assists in realizing the synergy of human and AI in SE and has been integrated into solution for SE problems, such as test generation (Marri et al. 2009; Taneja, Zhang, and Xie 2010; Thummalapenta et al. 2011), specification generation (Ammons, Bodk, and Larus 2002; Shambour 2019; Thummalapenta and Xie 2009), Debugging (Zeller 1999, 2009), and Programming (Gulwani 2010). Feldt, de Oliveira Neto, and Torkar (2018) proposed an initial taxonomy to categorize AI in SE application levels (AI-SEAL). AI-SEAL demonstrates the different ways of applying AI in SE and provides a prior knowledge for software engineer to consider the risks in utilizing AI.

In software quality context, artificial neural network (ANN)-based framework for software fault/defect prediction throughout SDLC phases has been proposed (Vashisht et al. 2015). The framework is interactive with developed graphical user interface. A historical dataset taken from 45 actual projects is involved in the experiments. The actual defect data are taken from completed projects on the basis of waterfall development model. For building the proposed framework, ANNs used this historical dataset as a training data in its training phase. Then, the developed network is used for defect prediction in all new projects. The input components for the network in the proposed framework consider production, prevention, rework, and review efforts. Users are required to enter the planned effort data of the five phases of SDLC. If the data provided by the user meets the given range of eligible criteria, then defects are estimated using the defect estimation system of the ANNs. The proposed framework is validated using 15 real-time projects, and the results show that actual defects lie inside the range of predicted defects. Moreover, the optimistic prediction quality and the quality of fit are introduced. The proposed framework is compared with evolutionary neural network (ENN) framework and linear regression-based prediction model. The proposed ANN framework has obtained up to 90% accuracy, whereas the ENN and linear regression frameworks have obtained 75% and 66% accuracy, respectively.

In the field of modeling and simulation, Fishwick (1992) constructed a broad multimodeling conceptual paradigm as an extension to the object-oriented modeling paradigm to address complexity by using different partial models at different scales and in

different operating contexts. One of the early frameworks is the use of a set of models to represent a single phenomenon and incorporate different abstraction levels. The framework attempts to unite the taxonomy and terminology of three disparate fields, namely, simulation, SE, and AI, to facilitate the interworking of these specialties. This methodology provides a way to structure a heterogeneous set of model types where each type performs its part, and the behavior is preserved while levels are mapped.

In computer science, AI and SE are contrasted and compared in terms of techniques and tools that they use, the methods they use, and the problems they attempt to solve. Barstow (1988) performed an excellent survey of the use of AI in SE. He distinguished between two categories of SE activities, that is, small- and large-scale programming. He categorized the knowledge used during software activities into five general categories, namely, methodologies of SE, programming techniques, the target machine's architecture, the amount of knowledge about the application domain, and the history of the target software. The author also described the roles played by this knowledge and argued that AI techniques can greatly assist with the handling of this knowledge. In other words, AI techniques can help manage the fundamental problems faced by SE effectively, thereby leading to arguments that effective knowledge management requires computer support and that computer support for knowledge management requires AI techniques. The author further argued that AI applied to SE has been relatively narrowly focused, whereas research on AI applied to SE has had virtually no demonstrations of practical value. He also claimed that the diversity and amount of the knowledge do not demonstrate practical success in this area in past research. For small-scale programming, some previous experimental systems, such as Mitre (Brown 1985) and AT&T (Kelly and Nonnenmann 1987), are good experimental systems in a practical situation.

AI techniques have introduced a significant potential for supporting and enhancing SE. Meziane and Vadera (2010) introduced a survey of some trends in using AI methods, such as genetic algorithms (GAs), neural networks, and natural language processing techniques, for the SDLC. In the software planning phase, GAs are the most commonly used technique. In practice, they are appropriate for adoption because they are flexible to represent different objectives and can easily represent schedules. ANNs, Bayesian networks, and case-based reasoning (CBR) proposed by Yang and Wang have been adopted for risk assessment with favor to Bayesian networks, which are more transparent and more appealing in practice than the other methods (Yang and Wang 2009). Knowledge-based systems (KBSs) are used to manage requirements and decisions taken during the design process. Ontologies have been used for requirements and design. Domain ontologies allow good understanding of the problem domain and detection of incompleteness and ambiguities because they encompass the strengths of KBS- and NLP-based systems. They also encapsulate knowledge and rules in a specific domain in a single resource. A number of researchers have attempted to use AI planning and GAs for generating test cases and suggested that using genetic programming can reduce the number of ill-defined test sequences. Moreover, using GAs for generating the order of integration of object-oriented classes is more promising than traditional graph-based methods. The review concluded that large-scale evaluation studies of AI techniques in SE are required. Thus, further research is needed to understand the effectiveness of different approaches.

In service-oriented architecture (SOA) design, Rodriguez et al. provided a detailed synthesized and conceptualized analysis of 69 significant studies that use AI approaches to discover, compose, or develop web services (Rodrguez, Soria, and Campo 2016). The review study attempted to answer some research questions, such as common features, major characteristics, and differences among the applied AI approaches in services. In terms of web service discovery process, among the studies conducted between 2002 and 2013 that have applied AI techniques, the most popular used AI approaches are ontologies, collaborative filtering, and information retrieval (IR). Ontologies are used because of their capability in machine-interpretable descriptions. Meanwhile, IR inherits a rich background. In terms of composing services, numerous frameworks and approaches have been developed to facilitate service composition. The widely used approaches between 2002 and 2015 in web service composition are planning techniques and evolutionary algorithms, which are used to build executable workflows of composed web services for meeting the service requirements. Depending on the composition environment, some AI techniques are

more appropriate than others. For instance, constraint optimization techniques are suitable for uncertain distributed and dynamic environments. AI planning is promising for dynamic web service compositions with incomplete information. For high-scalability requirements, evolutionary techniques are more useful than others. The third term is service development. CBR has been widely applied to achieve the essence of autonomic computing, such as availability, dependability, and robustness. CBR is suitable in well-defined application domains and in determining previous solutions to current problems with similar conditions. Moreover, it helps developers choose appropriate object-oriented designs to reify SOAs.

In SE practices, the experience of using AI domain has been reported for educating computer science students. Two large AI-based software systems have been successfully applied for instruction focusing on board game learning mechanisms and tree lifecycle management (Kalles 2016).

In (Poyet, DUBOIS, and Delcambre 1990), the authors presented two research projects conducted at the French Scientific and Technical Center for Building, which is involved in the area of AI applied to building engineering in a wide range of research programs. The review introduced software tools developed for suitable framework for knowledge modeling. As a result of the experience gained from the development of different systems, either based on design facilities, AI languages, large expert system, or restricted microcomputer environments, the authors created an object-based integrated environment that would facilitate data exchange, ensure machine independence, and take advantage of advanced SE policies, thereby further providing robustness and code saving and avoiding most pitfalls previously encountered, such as large tools in cumbersome design or basic languages in low-level modeling. A representative set of coupled systems, including object-oriented databases, were presented to provide immediately reusable information structures for remote processing, hypertext and hyperobject facilities linked to multifunctional expert systems, and database modeler that offers retrieval functions.

Raza (2009) highlighted the use of AI in solving SE problems to save time and effort during software development process. Applications of AI in expert system development and risk management were discussed. In expert system development, KBSs in AI assist traditional expert system approach. In addition, automated programming in AI is promising for reusable codes. Thus, when a certain part of a design is changed, the unchanged part will not be affected. Automated programming aims to minimize the specification, simplify the writing and understanding, and produce less error than programming languages. In risk management, automated programming makes data structures flexible, thereby making AI-based systems free from risk management strategies. Thus, the applying AI-based systems with the help of automated tools can save time and effort in software development and reduce risk assessment phase.

Computational intelligence (CI) serves as an algorithmic and conceptual framework to address the needs of knowledge-rich environment of SE. CI can handle data and knowledge coherently, thereby increasing assurance to SE. Pedrycz (2002) identified that CI is suitable to support SE activities. He linked three CI techniques, namely, granular computing, evolutionary optimization, and neural networks, to various SE activities, such as data visualization, evaluation of domain knowledge, and cost estimation. For example, the notion of information granules, such as rough sets, shadowed sets, probabilities, random sets, and fuzzy sets. SE plays a primary role in information granulation. The fundamentals of granular computing are compatible with the underlying paradigms of software products and process software. The role of information granules needs to be emphasized given project design details and testing plans. Meanwhile, neural networks, especially self-organizing maps, are useful for high-dimensional software data visualization.

Wooldridge (1997) intensively studied agent-based SE. He stated that ”agents are simply software components that must be designed and implemented in much the same way that other software components are. ” Software agents are encapsulated entities situated in a certain environment, aiming to achieve and meet their needs and design objectives and have superior flexibility and autonomy in that environment. The author highlighted the issues of building a software with respect to multi-agent-based system. A roadmap was set out in agent-based SE, where the considered fundamental issues of agent-based system were specification, implementation/refinement, and verification (including testing and debugging). The article discussed that software agent should

exhibit some principle characteristics, namely, reactive and proactive social behaviors. Thus, an agent should have the following key properties: (1) Autonomy: Agents are identifiable entities. They decide without any external intervention from other systems or humans. (2) Reactivity: Agents are embedded in a certain environment (such as a collection of other agents, physical world, the Internet, a user via a graphical user interface, or perhaps many of these combined). They can perceive this environment (at least to some extent) and react to changes in it. (3) Pro-activeness: Agents do not simply react to changes in the environment. They also exhibit goal-directed behavior by taking the initiative. (4) Social ability: Agents can cooperate with one another and engage in social activities to fulfill their design objectives. Therefore, an agent-embedded model is user-friendly, intuitive, adaptive, and flexible.

Tveit (2001) introduced some of the previous methodologies and applications of agent-oriented SE (AOSE) in real world; he highlighted design and high-level methodologies related to SE.

The term "intelligent" should be used because the software can have certain types of behavior, and the term "agent" pertains to the purpose of the software. An agent is "one who is authorized to act for or in the place of another" (Merriam Webster's Dictionary). The purposes of AOSE are to create tools and methodologies for agent-based software and enable low-cost development and maintenance. In addition, software should be high quality, scalable, easy to use, and flexible. Some examples of software agents include destructive agents, such as computer viruses, animated paperclip in MS Office, trading agents (such as auction agent in eBay), and Quake (an example of artificial player in games). For further details, the following points illustrate the types of high-level and design methodologies:

(1) High-level Methodologies

- *Gaia Methodology*:

Gaia is a general methodology that supports the macrolevel (that is, organization structure and agent society) and microlevel (namely, agent structure of agent development). Gaia addresses some limitations of existing methodologies where they fail to represent the nature of problem solving and autonomy of agents and model approaches for agents to create organizations and interact. Gaia allows software designers to systematically develop an implementation-ready design based on system requirements. Gaia is a good approach to develop closed-domain agent systems (Wooldridge, Jennings, and Kinny 2000, 1999).

- *The Multiagent Systems Engineering Methodology*:

MaSE is similar to Gaia with respect to generality and the application domain supported. However, MaSE goes further for using MaSE tool for automatic code creation. This methodology addresses the lack of industrial-strength toolkits and creates agent-based systems. MaSE aims to lead designers from the initial system specification to the implementation of agent-based system. MaSE has similar domain restrictions to those of Gaia. However, MaSE requires one-to-one agent interactions and not multicast (DeLoach 1999; Wood and DeLoach 2000).

- *Modeling database information systems*:

Agent-object relationship model aims to provide the capability to model relationship between agents in addition to static entities (Wagner 2003, 2001).

(2) Design Methods

- *UML*:

An architecture-centric design method for multiagent-based systems was presented by Yim et al. (2000). The method supports the transformation modeling problems from agent-oriented to object-oriented. In this method, developers and designers can use existing UML-based tools and experience and knowledge from developing object-oriented systems. A further extension to UML was proposed by Odell, Parunak, and Bauer (2000) and called it agent UML. The extension gives agents the ability to be mobile, that is, they can autonomously move among different agent-based systems. The application of four agent-oriented UML diagrams were suggested by Bergenti and Poggi (2000) applied at the highest abstraction level (agent level) of AOSE, where the standard of UML is not required to be changed.

- *Design Patterns*:

Design patterns are recurring patterns of programming code or component software architecture. In a mobile agent context, Aridor and Lange (1998) suggested a classification scheme for design patterns. The objectives are to increase quality and reusability of the code and reduce the development effort of mobile agent-based systems. Rana and Biancheri (1999) applied a Petri net-based

approach to model the meeting pattern of mobile agents. A seven-layer architecture pattern was proposed for agents, and sets of patterns belonged to each of the layers (Kendall, Malkoun, and Jiang 1997).

- *Components*:

Components are logical groups of related objects that can provide certain functionalities. Erol, Lang, and Levy (2000) suggested a three-tier architecture that applies reusable components to compose agents.

- *Graph Theory*:

Depke and Heckel et al. (2000) applied formal graph theory on requirement specifications for agent systems to maintain consistency when transferring the requirements into a design model.

4.3.3. Integration artificial intelligence and software engineering

The developments in AI and SE have several shared properties. They transact with modeling real-world objects, such as business rules, expert instruction, or process standards. A short survey was reported in (Rech and Althoff 2004) to provide an overview of these methods and explain any modern research topics on the framework of natural features of communication. A comprehensive review of the domain of combined AI and SE was introduced in (Partridge 1990). A taxonomy of this extension domain was explained and compared to other major efforts to solve the communication between the two domains. The three major subareas, namely, AI-based encouragement circumstances, AI tools and procedures in functional SE, and SE mechanisms and procedures in AI techniques, were explained and demonstrated with illustrative examples. Eventually, the domain of the extension should still be developed. Thus, any modern struggle to change the circumstances should be surveyed.

A tutorial introduction to AI techniques for SE developer and a similar introduction to SE development for AI techniques was introduced in (Ford 1987). These domains were analyzed and differentiated in terms of the difficulties they solve, the techniques they apply, and the utilized mechanisms and procedures. Merging the two domains is required for several different software requirements. The indication was reviewed briefly, and any of the actions required for an organization of the two domains were introduced.

As a model of utilizing the synergy in AI and SE domains, the domain of deep SE with AI was developed with several improvements in the previous years. Such domain broadly discusses problems on the intelligence of SE and the engineering of intelligence software. The recent and future studies concerning intelligent SE were introduced in (Xie 2018). The first problem, which is the intelligence of SE, converges the deep and intelligence strategies in methods to improve the performance and productivity in many different SE jobs. The second problem, which is the engineering of intelligence software, converges many different SE jobs for intelligent and deep software (such as AI software).

SE and AI are the major two domains in computer science. During the recent years, the methods of these domains have been improved individually with no considerable interchange of research results. However, both domains have various features, attributes, benefits, and limitations. This statement presents several chances and plans for new studies. A major important idea is that the researcher utilizes the possible methods, instruments, and methods of AI to SE, and vice versa in practice. In this manner, knowledge, experiences, characteristics, and resources of both fields can be collected, and the controls can decrease. Regarding applicability, a junction domain is located between AI and SE, thereby establishing the relationship between AI and SE. The factors during the interaction between AI and SE include information, goal, difficulty, and motivations for using. The framework of cooperation on which both domains interact with each other was explored in (Jain 2011). This framework contains four major types of interaction, namely, software maintenance setting, AI tools and techniques in traditional software, the performance of conventional software technology, and methodological problems. This paper presented the connection between AI and SE and several systems developed while integrating both domains.

5. Discussion

This section highlights two major points based on the above survey findings. First, as illustrated in Figure 4, the taxonomy of this review shows that the usage of

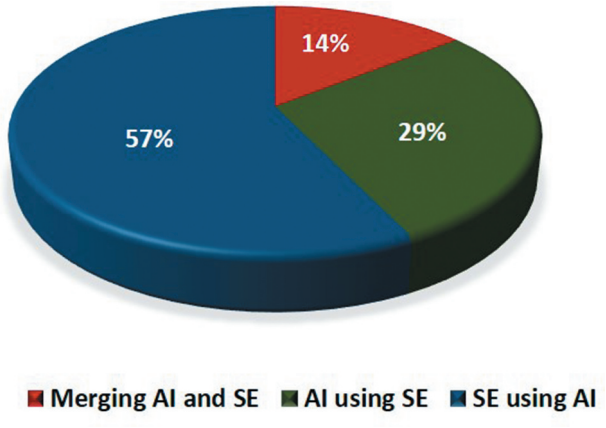

Figure 4. The distribution of published research articles.

SE for the techniques, methods, or tools of AI reached 57% compared with the usage of AI for those of SE (that is, SDLC), which reached 29%. Meanwhile, the integration of AI and SE reached 14%. These percentages are logical because AI contains many subfields, such as optimization algorithms and machine learning. Therefore, software engineers have many choices to improve the performance of their works, as mentioned in Section 4.2.

Second, new versions of the software are generated by integrating AI and SE, which leads to many possibilities. Figure 5 shows an overview of each field with the interaction area between them. AI includes knowledge acquisition, domain modeling, and data analysis techniques. SE contains project management methods, requirements engineering, and code engineering. The intersection area between AI and SE includes KBS, AOSE, CI, and ambient intelligence.

6. Conclusion and future works

This review focuses on the relationships between AI and SE. It highlights the effect of AI on the performance of SE, and vice versa. The overview of each AI and SE, as well as their definitions, tools, features, and weaknesses, are provided. This overview helps the researchers in each field to understand the other fields. The second stage illustrates the growth of the integration between AI and SE, the benefits of the integration, and the presentation of the previous works. The conclusion shows several research fields on AI and SE, where the percentages of using the research on AI in SE, SE in AI, and integration of both are presented. The interaction between both domains is also presented. In future works, the techniques and methods for each field will be considered, and their effects will be studied. For instance, machine learning from AI and the requirements in the SE will be covered.

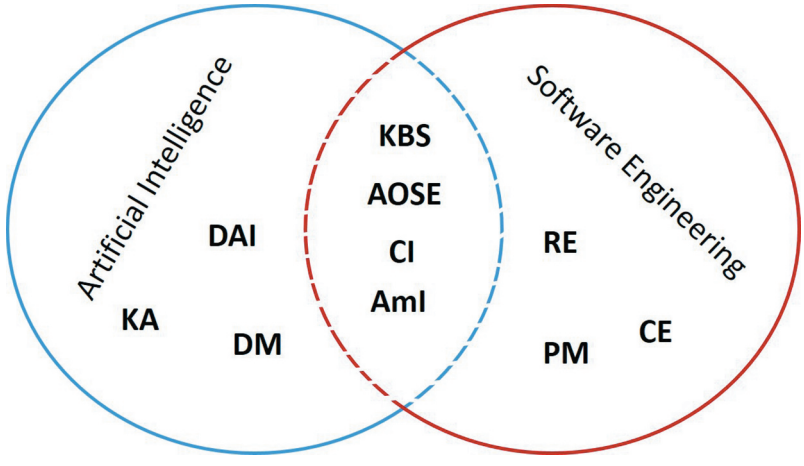

Figure 5. Interaction between AI and SE.

Disclosure statement

The authors declare that they have no conflicts of interest.

ORCID

Mohammad Shehab http://orcid.org/0000-0003-0211-3503

References

Abad, Z. S. H., M. Noaeen, and G. Ruhe. 2016. "Requirements Engineering Visualization: A Systematic Literature Review." In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, 6–15. Beijing, China: IEEE.

Abualigah, L., M. Shehab, M. Alshinwan, and H. Alabool. 2019. "Salp Swarm Algorithm: A Comprehensive Survey." *Neural Computing & Applications*31: 1–21.

Abualigah, L., M. Shehab, M. Alshinwan, S. Mirjalili, and M. Abd Elaziz. 2020. "Ant Lion Optimizer: A Comprehensive Survey of Its Variants and Applications." *Archives of Computational Methods in Engineering*. doi:10.1007/s11831-020-09420-6.

Abu-Hashem, M. A., N. A. Rashid, R. Abdullah, A. A. Hasan, and A. A. Abdulrazzaq. 2015. "Investigation Study: An Intensive Analysis for Msa Leading Methods." *Journal of Theoretical & Applied Information Technology* 75 (1): 1–12.

Abu-Hashem, M. A., D. M. Uliyan, and A. Abuarqoub. 2017. "A Shared Memory Method for Enhancing the Htngh Algorithmperformance: Proposed Method." In *Proceedings of the International Conference on Future Networks and Distributed Systems*, 14. Cambridge, UK: ACM.

Aleem, S., L. F. Capretz, and F. Ahmed. 2016. "Game Development Software Engineering Process Life Cycle: A Systematic Review." *Journal of Software Engineering Research and Development* 4 (1): 6. doi:10.1186/s40411-016-0032-7.

Al-Zewairi, M., M. Biltawi, W. Etaiwi, and A. Shaout. 2017. "Agile Software Development Methodologies: Survey of Surveys." *Journal of Computer and Communications* 5 (5): 74–97. doi:10.4236/jcc.2017.55007.

Ammar, H. H., W. Abdelmoez, and M. S. Hamdi 2012. "Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems." In *Proceedings of the First Taibah University International Conference on Computing and Information Technology (ICCIT 2012), Al-Madinah Al-Munawwarah, Saudi Arabia*, 52.

Ammons, G., R. Bodk, and J. R. Larus. 2002. "Mining Specifications." *ACM Sigplan Notices* 37 (1): 4–16. doi:10.1145/565816.503275.

Aridor, Y., and D. B. Lange. 1998. "Agent Design Patterns: Elements of Agent Application Design." In *Agents*, 108–115. Vol. 98. USA: ACM.

Barstow, D. 1988. "Artificial Intelligence and Software Engineering." In *Exploring Artificial Intelligence*, 641–670. San Francisco, California: Elsevier.

Bergenti, F., and A. Poggi. 2000. "Exploiting Uml in the Design of Multi-agent Systems." In *International Workshop on Engineering Societies in the Agents World*, 106–113. Berlin, Germany: Springer.

Boehm, B. W., and R. Turner. 2015. "The Incremental Commitment Spiral Model (Icsm): Principles and Practices for Successful Systems and Software." In *ICSSP*, 175–176. University of Southern California.

Bonati, C., E. Calore, S. Coscetti, M. D'elia, M. Mesiti, F. Negro, S. F. Schifano, and R. Tripiccione. 2015. "Development of Scientific Software for Hpc Architectures Using Openacc: The Case of Lqcd." In *Proceedings of the 2015 International Workshop on Software Engineering for High Performance Computing in Science*, 9–15. Florence, Italy: IEEE Press.

Brown, R. 1985. "Automation of Programming; the Isfi Experiments." In *Proc. Of Expert Systems in Government Symposium*, 525–539. Mclean.

Chakraborty, P., R. Shahriyar, A. Iqbal, and A. Bosu. 2018. "Understanding the Software Development Practices of Blockchain Projects: A Survey." In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 28. Oulu, Finland: ACM.

Chapman, W. 2018. *Engineering Modeling and Design*. Vol. 33. Routledge.

Chowdhury, M., and A. W. Sadek. 2012. "Advantages and Limitations of Artificial Intelligence." *Artificial Intelligence Applications to Critical Transportation Issues* 6 (3): 360–375.

Claypool, K., and M. Claypool. 2005. "Teaching Software Engineering through Game Design." In *ACM SIGCSE Bulletin*. Vol. 37, 123–127. ACM.

DeLoach, S. A. 1999. "Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems." Technical report, air force inst of tech wright-patterson afb oh dept of electrical.

Depke, R., R. Heckel, et al. 2000. "Formalizing the Development of Agent-based Systems Using Graph Processes." In *ICALP Satellite Workshops*, 419–426. Paderborn, Germany.

Devadiga, N. M. 2017. "Tailoring Architecture Centric Design Method with Rapid Prototyping." In *2017 2nd International Conference on Communication and Electronics Systems (ICCES)*, 924–930. Monte De Caparica, Portuga: IEEE.

Eisty, N. U., G. K. Thiruvathukal, and J. C. Carver. 2019. "Use of Software Process in Research Software Development: A Survey." 3 (3): 180–202.

Erol, K., J. Lang, and R. Levy. 2000. "Designing Agents from Reusable Components." In *Proceedings of the Fourth International Conference on Autonomous Agents*, 76–77. Barcelona, Spain: ACM.

Falzone, E., and C. Bernaschina. 2018. "Model Based Rapid Prototyping and Evolution of Web Application." In *International Conference on Web Engineering*, 496–500. Cáceres, Spain: Springer.

Feldt, R., F. G. de Oliveira Neto, and R. Torkar. 2018. "Ways of Applying Artificial Intelligence in Software Engineering." In *Proceedings of the 6th International Workshop on Realizing*

Artificial Intelligence Synergies in Software Engineering, 35–41. Gothenburg, Sweden: ACM.

Fink, L., S. Wyss, and Y. Lichtenstein. 2018. "Aligning Flexibility with Uncertainty in Software Development Arrangements through a Contractual Typology." *Journal of Global Operations and Strategic Sourcing* 11 (1): 2–26. doi:10.1108/JGOSS-11-2016-0033.

Fishwick, P. A. 1992. "An Integrated Approach to System Modeling Using a Synthesis of Artificial Intelligence, Software Engineering and Simulation Methodologies." *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 2 (4): 307–330. doi:10.1145/149516.149530.

Ford, L. 1987. "Artificial Intelligence and Software Engineering: A Tutorial Introduction to Their Relationship." *Artificial Intelligence Review* 1 (4): 255–273. doi:10.1007/BF00142926.

Gui, J., S. Mcilroy, M. Nagappan, and W. G. Halfond. 2015. "Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers." In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, 100–110. Florence, Italy: IEEE Press.

Gulwani, S. 2010. "Dimensions in Program Synthesis." In *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, 13–24. Hagenberg, Austria: ACM.

Hamet, P., and J. Tremblay. 2017. "Artificial Intelligence in Medicine." *Metabolism* 69: S36–S40. doi:10.1016/j.metabol.2017.01.011.

Harman, M. 2012. "The Role of Artificial Intelligence in Software Engineering." In *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*, 1–6. Zurich, Switzerland: IEEE.

Hassan, M. M., W. Afzal, M. Blom, B. Lindström, S. F. Andler, and S. Eldh. 2015. "Testability and Software Robustness: A Systematic Literature Review." In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, 341–348. Funchal, Portugal: IEEE.

Jain, P. 2011. "Interaction between Software Engineering and Artificial Intelligence-a Review." *International Journal on Computer Science and Engineering* 3 (12): 3774.

Jain, P., A. Sharma, and L. Ahuja. 2018. "Software Maintainability Estimation in Agile Software Development." *International Journal of Open Source Software and Processes (IJOSSP)* 9 (4): 65–78. doi:10.4018/IJOSSP.2018100104.

Johanson, A., and W. Hasselbring. 2018. "Software Engineering for Computational Science: Past, Present, Future." *Computing in Science & Engineering* 11 (3): 52–66.

Kalles, D. 2016. "Artificial Intelligence Meets Software Engineering in Computing Education." In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence*, 36. Thessaloniki, Greece: ACM.

Kelly, V. E., and U. Nonnenmann. 1987. "Inferring Formal Software Specifications from Episodic Descriptions." In *Sixth National Conference on Artificial Intelligence*, 127-132. Seattle, Washington.

Kendall, E. A., M. T. Malkoun, and C. H. Jiang. 1997. "The Application of Object-oriented Analysis to Agent-based Systems." *JOOP* 9 (9): 56–62.

Kramer, M. 2018. "Best Practices in Systems Development Lifecycle: An Analyses Based on the Waterfall Model." *Review of Business & Finance Studies* 9 (1): 77–84.

Krishnan, M. S. 2015. "Software Development Risk Aspects and Success Frequency on Spiral and Agile Model." *International Journal of Innovative Research in Computer and Communication Engineering* 5 (3): 1.

Kuhrmann, M., P. Diebold, J. Münch, P. Tell, V. Garousi, M. Felderer, K. Trektere, F. McCaffery, O. Linssen, E. Hanser, et al. 2017. "Hybrid Software and System Development in Practice: Waterfall, Scrum, and Beyond." In *Proceedings of the 2017 International Conference on Software and System Process*, 30–39. Paris, France: ACM.

Kurzweil, R., R. Richter, R. Kurzweil, and M. L. Schneider. 1990. *The Age of Intelligent Machines*, 579. Cambridge, MA: MIT press.

Ma, L., F. Juefei-Xu, M. Xue, Q. Hu, S. Chen, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See. 2018. "Secure Deep Learning Engineering: A Software Quality Assurance Perspective." *arXiv Preprint arXiv:1810.04538* 3 (1): 42–54.

Malhotra, R., and A. J. Bansal. 2016. "Software Change Prediction: A Literature Review." *International Journal of Computer Applications in Technology* 54 (4): 240–256. doi:10.1504/IJCAT.2016.080487.

Marri, M. R., T. Xie, N. Tillmann, J. De Halleux, and W. Schulte. 2009. "An Empirical Study of Testing File-system-dependent Software with Mock Objects." In *2009 ICSE Workshop on Automation of Software Test*, 149–153. Vancouver, BC, Canada: IEEE.

Meja Niño, C., M. Albano, E. Jantunen, P. Sharma, J. Campos, and D. Baglee. 2018. "An Iterative Process to Extract Value from Maintenance Projects." In *3 Conferência Internacional Sobre Engenharia De Manutenção (Income-iii 2018)*, 319–335. Coimbra, Portugal: APMI.

Menghi, C., A. M. Rizzi, and A. Bernasconi. 2018. "Integrating Topological Proofs with Model Checking to Instrument Iterative Design." *arXiv Preprint arXiv:1811.11123* 21 (3): 249–259.

Meziane, F., and S. Vadera. 2010. "Artificial Intelligence in Software Engineering: Current Developments and Future Prospects." In *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects*, 278–299. Andrews, UK: IGI Global.

Mistry, S. V. 2017. "Homelessness and Trust: The Effects of Homeless Intake Verification on Relationships." *Washington and Lee University*33 (2): 365–374.

Noureddine, A., R. Rouvoy, and L. Seinturier. 2015. "Monitoring Energy Hotspots in Software." *Automated Software Engineering* 22 (3): 291–332. doi:10.1007/s10515-014-0171-1.

Odell, J., H. V. D. Parunak, and B. Bauer. 2000. "Extending Uml for Agents." In *Proceedings of the Agent-oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence*, 3–17. Austin, USA.

Partridge, D. 1990. "Artificial Intelligence and Software Engineering: A Survey of Possibilities." In *The Software Life Cycle*, 375–385. Elsevier.

Partridge, D. 1992. *Engineering Artificial Intelligence Software*. Vol. 9. Oxford and Norwood: Intellect Books.

Pedrycz, W. 2002. "Computational Intelligence as an Emerging Paradigm of Software Engineering." In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, 7–14. Ischia, Italy: ACM.

Poyet, P., A.-M. DUBOIS, and B. Delcambre. 1990. "Artificial Intelligence Software Engineering in Building Engineering." *Computer-Aided Civil and Infrastructure Engineering* 5 (3): 167–205. doi:10.1111/j.1467-8667.1990.tb00376.x.

Qiang, B., and E. A. Peña. 2018. "Improved Estimation of System Reliability with Application in Software Development." *Analytic Methods in Systems and Software Testing* 17 (5): 255–275.

Räihä, O. 2010. "A Survey on Search-based Software Design." *Computer Science Review* 4 (4): 203–249. doi:10.1016/j.cosrev.2010.06.001.

Ramamoorthy, C. V., and Y.-C. Shim. 1991. "On Issue in Software Engineering and Artificial Intelligence." *International Journal of Software Engineering and Knowledge Engineering* 2 (1), 8–15.

Rana, O. F., and C. Biancheri. 1999. "A Petri Net Model of the Meeting Design Pattern for Mobile-stationary Agent Interaction." In *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32. Abstracts and CD-ROM of Full Papers*, 9–16. Maui, HI, USA: IEEE.

Rao, P. V., V. P. Kumar, and B. P. K. Reddy. 2018. "Applying Agile Software Methodology for the Development of Software Development Life Cycle Process (Sdlc)." *Journal for Research— Volume* 4 (2): 125–136.

Raza, F. N. 2009. "Artificial Intelligence Techniques in Software Engineering (Aitse)." In *International MultiConference of Engineers and Computer Scientists (IMECS 2009)*, 10–17. Vol. 1. Hong Kong.

Rech, J., and K.-D. Althoff. 2004. "Artificial Intelligence and Software Engineering: Status and Future Trends." *KI* 18 (3): 5–11.

Ringert, J. O., B. Rumpe, C. Schulze, and A. Wortmann. 2017. "Teaching Agile Model-driven Engineering for Cyber-physical Systems." In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*, 127–136. Buenos Aires, Argentina: IEEE Press.

Rodrguez, G., Á. Soria, and M. Campo. 2016. "Artificial Intelligence in Service-oriented Software Design." *Engineering Applications of Artificial Intelligence* 53 (3): 86–104. doi:10.1016/j.engappai.2016.03.009.

Santhanam, G. R. 2016. "Qualitative Optimization in Software Engineering: A Short Survey." *Journal of Systems and Software* 111: 149–156. doi:10.1016/j.jss.2015.09.001.

Semeráth, O., A. Vörös, and D. Varró. 2016. "Iterative and Incremental Model Generation by Logic Solvers." In *International Conference on Fundamental Approaches to Software Engineering*, 87–103. Eindhoven, The Netherlands: Springer.

Shahkarami, A., S. D. Mohaghegh, V. Gholami, S. A. Haghighat, et al. 2014. "Artificial Intelligence (Ai) Assisted History Matching." In *SPE Western North American and Rocky Mountain Joint Meeting*, 369–381. Denver, Colorado: Society of Petroleum Engineers.

Shambour, M. K. Y. 2017. "Dynamic Search Zones (Dsz) for Harmony Search Algorithm." In *2017 8th International Conference on Information Technology (ICIT)*, 941–946.

Shambour, M. K. Y. 2019. "Adaptive Multi-crossover Evolutionary Algorithm for Real-world Optimisation Problems." *International Journal of Reasoning-based Intelligent Systems* 11 (1): 1–10. doi:10.1504/IJRIS.2019.098058.

Shambour, M. K. Y., A. A. Abusnaina, and A. I. Alsalibi. 2019. "Modified Global Flower Pollination Algorithm and Its Application for Optimization Problems." *Interdisciplinary Sciences, Computational Life Sciences* 11 (3): 496–507. doi:10.1007/s12539-018-0295-2.

Shambour, Y., et al. 2018. "Vibrant Search Mechanism for Numerical Optimization Functions." *Journal of Information & Communication Technology* 17 (4): 679–702.

Shankari, K. H., and R. Thirumalaiselvi. 2014. "A Survey on Using Artificial Intelligence Techniques in the Software Development Process." *International Journal of Engineering Research and Applications* 4 (12): 24–33.

Shehab, M. 2020a. *Adaptive Cuckoo Search Algorithm for Extracting the ODF Maxima*, 77–89. Switzerland: Springer International Publishing.

Shehab, M. 2020b. *Modified Cuckoo Search Algorithm (MCSA) for Extracting the ODF Maxima*, 91–110. Switzerland: Springer International Publishing.

Shehab, M., L. Abualigah, H. Al Hamad, H. Alabool, M. Alshinwan, and A. M. Khasawneh. 2019. "Moth–flame Optimization Algorithm: Variants and Applications." *Neural Computing & Applications*. doi:10.1007/s00521-019-04570-6.

Shehab, M., H. Alshawabkah, L. Abualigah, and A.-M. Nagham. 2020. "Enhanced a Hybrid Moth-flame Optimization Algorithm Using New Selection Schemes." *Engineering with Computers* 36: 1–26.

Shehab, M., A. T. Khader, and M. A. Al-Betar. 2017. "A Survey on Applications and Variants of the Cuckoo Search Algorithm." *Applied Soft Computing* 61: 1041–1059. doi:10.1016/j.asoc.2017.02.034.

Stark, G. E., P. Oman, A. Skillicorn, and A. Ameele. 1999. "An Examination of the Effects of Requirements Changes on Software Maintenance Releases." *Journal of Software Maintenance: Research and Practice* 11 (5): 293–309. doi:10.1002/(SICI)1096-908X(199909/10)11:5<293::AID-SMR198>3.0.CO;2-R.

Stewart, J. 2015. "Strong Artificial Intelligence and National Security: Operational and Strategic Implications." Technical report, naval war college newport ri joint military operations dept.

Tahir, M., F. Khan, M. Babar, F. Arif, and F. Khan. 2016. "Framework for Better Reusability in Component Based Software Engineering." *The Journal of Applied Environmental and Biological Sciences (JAEBS)* 6 (4S): 77–81.

Tamalika, B., B. Avantika, D. Joseph, and Ramanathan. 2017. "A Survey on the Role of Artificial Intelligence in Software Engineering." *International Journal of Innovative Research in Computer and Communication Engineering* 5 (4): 7062–7066.

Taneja, K., Y. Zhang, and T. Xie. 2010. "Moda: Automated Test Generation for Database Applications via Mock Objects." In

Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, 289–292. Antwerp, Belgium: ACM.

Tenne, Y., and C.-K. Goh. 2010. *Computational Intelligence in Expensive Optimization Problems*. Vol. 2. Switzerland: Springer Science & Business Media.

Thummalapenta, S., and T. Xie. 2009. "Alattin: Mining Alternative Patterns for Detecting Neglected Conditions." In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, 283–294. Auckland, New Zealand: IEEE Computer Society.

Thummalapenta, S., T. Xie, N. Tillmann, J. De Halleux, and Z. Su. 2011. "Synthesizing Method Sequences for High-coverage Testing." In *ACM SIGPLAN Notices*. Vol. 46, 189–206. Portland, Oregon, USA: ACM.

Tunio, M. Z., H. Luo, C. Wang, F. Zhao, W. Shao, and Z. H. Pathan. 2018. "Crowdsourcing Software Development: Task Assignment Using Pddl Artificial Intelligence Planning." *Journal of Information Processing Systems* 14 (1): 129–139.

Tveit, A. 2001. "A Survey of Agent-oriented Software Engineering." In *NTNU Computer Science Graduate Student Conference, Norwegian University of Science and Technology*.

Varró, D., G. Bergmann, Á. Hegedüs, Á. Horváth, I. Ráth, and Z. Ujhelyi. 2016. "Road to a Reactive and Incremental Model Transformation Platform: Three Generations of the Viatra Framework." *Software & Systems Modeling* 15 (3): 609–629. doi:10.1007/s10270-016-0530-4.

Vashisht, V., M. Lal, G. Sureshchandar, and S. Kamya. 2015. "A Framework for Software Defect Prediction Using Neural Networks." *Journal of Software Engineering and Applications* 8 (8): 384. doi:10.4236/jsea.2015.88038.

Wagner, G. 2001. "Agent-oriented Analysis and Design of Organisational Information Systems." In *Databases and Information Systems*, 111–124. Switzerland. Janis Barzdins, Albertas Caplinskas: Springer.

Wagner, G. 2003. "The Agent–object-relationship Metamodel: Towards a Unified View of State and Behavior." *Information Systems* 28 (5): 475–504. doi:10.1016/S0306-4379(02)00027-3.

Winston, P. H., and K. A. Prendergast. 1984. *The AI Business: Commercial Uses of Artificial Intelligence*. Vol. 25. Cambridge, MA United States: Massachusetts Institute of Technology.

Winter, E., S. Forshaw, and M. A. Ferrario. 2018. "Measuring Human Values in Software Engineering." In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 48. Oulu, Finland: ACM.

Wood, M. F., and S. A. DeLoach. 2000. "An Overview of the Multiagent Systems Engineering Methodology." In *International Workshop on Agent-Oriented Software Engineering*, 207–221. Limerick, Ireland: Springer.

Wooldridge, M. 1997. "Agent-based Software Engineering." *IEE Proceedings-software* 144 (1): 26–37. doi:10.1049/ip-sen:19971026.

Wooldridge, M., N. R. Jennings, and D. Kinny. 1999. "A Methodology for Agent-oriented Analysis and Design." In *Proceedings of the third annual conference on Autonomous Agents*, 69-76.

Wooldridge, M., N. R. Jennings, and D. Kinny. 2000. "The Gaia Methodology for Agent-oriented Analysis and Design." *Autonomous Agents and Multi-agent Systems* 3 (3): 285–312. doi:10.1023/A:1010071910869.

Xie, T. 2013. "The Synergy of Human and Artificial Intelligence in Software Engineering." In *2013 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, 4–6. San Francisco, CA, USA: IEEE.

Xie, T. 2018. "Intelligent Software Engineering: Synergy between Ai and Software Engineering." In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, 3–7. Beijing, China: Springer.

Yampolskiy, R. V., and M. Spellchecker. 2016. "Artificial Intelligence Safety and Cybersecurity: A Timeline of Ai Failures." *arXiv Preprint arXiv:1610.07997* 14 (6): 344–360.

Yang, H.-L., and C.-S. Wang. 2009. "Recommender System for Software Project Planning One Application of Revised Cbr Algorithm." *Expert Systems with Applications* 36 (5): 8938–8945. doi:10.1016/j.eswa.2008.11.050.

Yim, H., K. Cho, J. Kim, and S. Park. 2000. "Architecture-centric Object-oriented Design Method for Multi-agent Systems." In *Proceedings Fourth International Conference on MultiAgent Systems*, 469–470. Boston, MA, USA: IEEE.

Zeller, A. 1999. "Yesterday, My Program Worked. Today, It Does Not. Why?" In *ACM SIGSOFT Software Engineering Notes*. Vol. 24, 253–267. Passau, Germany: Springer-Verlag.

Zeller, A. 2009. *Why Programs Fail: A Guide to Systematic Debugging*. Vol. 22. Dagstuhl, Germany: Elsevier.

Zhang, Y., A. Finkelstein, and M. Harman. 2008. "Search Based Requirements Optimisation: Existing Work and Challenges." In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 88–94. Montpellier, France: Springer.

Zohair, L. M. A. 2018. "The Future of Software Engineering by 2050s: Will Ai Replace Software Engineers?" *International Journal of Information Technology and Language Studies* 2 (3).