AI Perspectives

RESEARCH ARTICLE Open Access

Applications of AI in classical software engineering

Marco Barenkamp[1*], Jonas Rebstadt[2] and Oliver Thomas[2]

Abstract

Although Artificial Intelligence (AI) has become a buzzword for self-organizing IT applications, its relevance to software engineering has hardly been analyzed systematically. This study combines a systematic review of previous research in the field and five qualitative interviews with software developers who use or want to use AI tools in their daily work routines, to assess the status of development, future development potentials and equally the risks of AI application to software engineering. The study classifies the insights in the software development life cycle. The analysis results that major achievements and future potentials of AI are a) the automation of lengthy routine jobs in software development and testing using algorithms, e.g. for debugging and documentation, b) the structured analysis of big data pools to discover patterns and novel information clusters and c) the systematic evaluation of these data in neural networks. AI thus contributes to speed up development processes, realize development cost reductions and efficiency gains. AI to date depends on man-made structures and is mainly reproductive, but the automation of software engineering routines entails a major advantage: Human developers multiply their creative potential when using AI tools effectively.

Keywords: Software engineering, AI, Artificial intelligence, ML, Machine learning, Deep learning, Pattern recognition, Neural networks

Introduction

Artificial intelligence has become a buzzword in popular and academic media. The prophecies and futuristic legends connected to artificial intelligence are multiple and well known: Computers will take over classical human engineering and development jobs [1], could even fully substitute human productivity by intelligent automation [2] and - in the worst case – govern a machine dominated brave new world [3]. In such scenarios classical software engineers would possibly become obsolete since machines could take over their tasks. Software would possibly not require any external engineering any more but develop self-reliantly [4–6]. Karpathy (2017) calls neural networks as a “software 2.0” which in the long run will supersede classical programming, like Java or C++, which mainly done by humans: AI could enable computers to self-reliantly produce code and solve problems [7].

To date, however - as compared to these future visions - artificial intelligence is in its beginnings, which shines up when searching for a recognized definition of the term. Intelligent biological and ideally intelligent artificial systems (e.g. machines) dispose of the ability to “think and learn” [8, 9]. Differential psychology distinguishes intelligence - as rational reflection - from other forms of mental activity, like emotionality and creativity. Factor models analyzing intelligent behavior count spatial perception, numerical abilities, memory, reasoning as well as verbal expression and interpretation among intelligent mental abilities [10, 11].

Artificial intelligence today is an umbrella term used for a set of computer-based routines which approximate human intelligence in the way that alternatives are weighed, new information is considered and integrated into existing data structures and new conclusions are

* Correspondence: marco.barenkamp@lmis.de
Potential applications of Artificial Intelligence for enhancing the efficiency of classical Software Engineering
[1]LMIS AG, Osnabrück, Germany
Full list of author information is available at the end of the article

reached by inference from qualitative or quantitative data or probabilistic estimates [12].

AI routines differ in their self-reliance and level of automation i.e. in the extent to which they require human support or ask for human feedback before they implement decisions or information [12]. They use a series of technologies. Among the most important are big data analytics, machine learning and more specifically, artificial neural networks [13, 14]:

- Big data analytics retrieves large amounts of data from diverse sources and evaluates these using particular queries and statistical evaluation routines [15]. Artificial intelligence automates information gathering and evaluation [16].
- Machine learning is a method of data analysis directed to identify patterns in unstructured data sets, which enables machines to draw conclusions and take decisions based on these classifications [17].
- Artificial neural networks frequently comprise several layers of mathematical routines which collect, classify and arrange data into new sets in order to find "correct" parameters or solutions. Neural network based deep learning is an approach of information integration and selection across several logical layers of an electronic information network [18, 19]. In that process, large data sets are repeatedly evaluated and interconnected using statistical and probabilistic routines [20] to generate comprehensive and systematic information and decision frameworks [21]. Algorithms are used to train neutral networks (backpropagation, variants of gradient descent). Neural networks can be distinguished by the type of data that they use during training or test time (labeled, unlabeled, categorical, numerical), their loss/error/cost/objective function, their connection patterns, and their optimization algorithm.

Applying these technologies artificial intelligence manages complex tasks like natural language processing [22], i.e. the understanding and translation of human language into other languages and codes, or computer vision, i.e. the visual perception, analysis and understanding of optical environmental information [23].

Classifications of AI in software engineering in earlier review-based studies

At first sight, AI application in software engineering seems a contradiction in terms since artificial intelligence is about routinizing operations by relying on "intelligent machines" while software engineering, is a creative and knowledge intensive activity, usually involving human experts. At second sight however, creative processes in software engineering could be effectively supported by machines, which backed by self-optimizing algorithms take over organization and optimization tasks [24]. Artificial intelligence, understood as machine-based perception, reasoning and interpretation of environmental and mental constructs [25, 26], could multiply the approaches and strategies of software engineering, i.e. the systematic approach to the analysis, design, assessment, implementation, test, maintenance and reengineering of software [27].

To discuss the potentials and limitations of AI for software engineering, categories describing the impacts of AI is useful. Previous reviews in the field use various classifications, which are systematized as object-related, function-related or process related in the following:

Object-related definitions of AI refer to the (usually complex) performance output artificial intelligence provides: Lu et al. [28] explain the relevance of AI for robot, automated driving, information and communication technology. AI equally supports game programmers [29, 30] and language experts in the development of translation programs and computer languages [31–34]. Classifying AI technologies by the output they generate entails the difficulty that diverse AI functionalities interact to produce a certain technical result. These are usually hard to understand and to differentiate for outsiders not involved in the development of the software and coding process.

Function-related classifications of AI in software engineering refer to the technologies of AI applied in the software engineering process. Diverse function-related classifications of AI have been suggested. Most earlier review-based studies with a technical focus extract certain fields of AI application in software engineering from earlier more specific discussions, which are evaluated qualitatively or quantitatively: Jarrahi [35] for examples sees the main potentials of AI in software engineering in language processing i.e. the interpretation and recognition of human language), machine learning (the analysis and adoption of work flows) and machine vision (strategic and target focused machine problem solution). Similarly, Muenchaisri et al. [13] explain the AI functions, e.g. machine learning, neural network, and natural language processing. Savchenko's et al. [14] review of 54 studies identifies the fields of big data and IoT technologies, programming and design assistance tools, machine learning, knowledge management and recommendation tools and system analysis tools, based on a more systematic quantitative classification of previous studies. Function-related classifications of AI in software engineering are empirically founded but risk that so far undiscovered domains, which AI could plausibly support, are neglected, since these are not yet explored empirically.

Similarly, limitations of AI in particular domains could be overlooked if no empirical analyses on these problems are available.

Process-related classifications of AI in software engineering refer to the stages of software development and analyze to what extent AI can support software engineering at the respective stage. This approach is apt to prevent the bias of neglecting AI limitations and development requirements, functional classifications are subject to. Only two reviews using process related classifications have been identified: Sorte et al. [36] and Padmanaban et al. [37] suggest to classify AI applications according to three stages of the software engineering life cycle: the software development, testing and deployment and maintenance phase. Their analyses however lack topical sources, systematic data research and in result are incomplete concerning the discussion of opportunities and limitations of the technologies. They are rather explorative than systematic.

Methodology

Research contributions

This study systematizes and empirically founds Padmanaban al.'s [37] and Sorte et al.'s [36] approach of classifying applications of AI for software engineering based on the software engineering life cycle model. It amends on earlier review-based research in three crucial points:

1. It intends a more differentiated discussion of the impact of AI on a more comprehensive model of the software development life cycle (compare Fig. 1).
2. As compared to earlier studies on the relevance of AI in software engineering, which are mainly focused on the potentials of the technology, this study equally discusses possible limitations based on a technical appraisal of available AI technologies.
3. Objectives 1 and 2 require a more comprehensive method than a pure review: Earlier reviews in AI in software engineering are incomprehensive concerning the role of AI across the software development life cycle, since they focus on individual technologies.

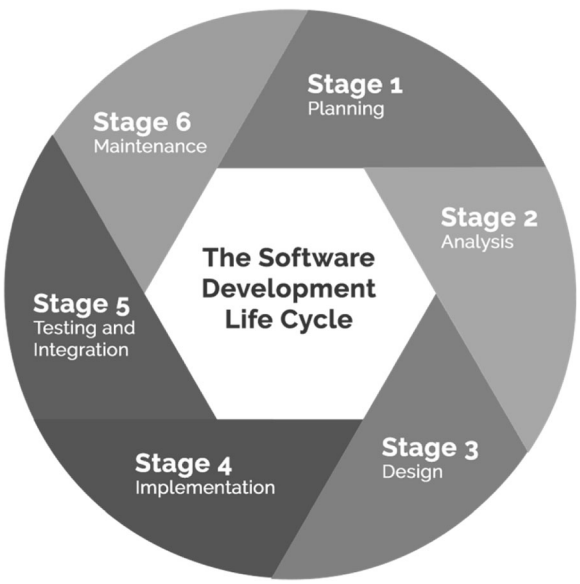

Fig. 1 Software Development Life Cycle (illustration adapted from [38])

This study uses a mixed method approach to realize a more comprehensive perspective. It combines a review of more than 60 previous studies in the field and own empirical expert interviews. The exact methodology is described in section 3.3.

Theoretical framework

To implement the review a six-stages-model of the software engineering life cycle is referred to, which has found broad application in academic software engineering literature [39–41] and software development practice [42, 43]. It comprises the stages of project planning, problem analysis, software design, implementation in software code, software testing and integration and software support and maintenance (Fig. 1).

Mixed-method approach

To assess the opportunities, limitations and potential risks of AI applications at the six stages of the life cycle comprehensively the study relies on two data sources – a systematic review and qualitative expert interviews. This combined qualitative research strategy is useful for issues which are at an early stage of systematic exploration, particularly, when established categories are not yet available, but still have to be developed [44]:

The systematic review of prior empirical studies explicitly refers to experiences with AI application at the respective stages of the development life cycle. The review includes more than 60 publications in peer-reviewed journals and conference papers published between 2010 and 2020, to ensure topicality and academic quality of the results. The review results are evaluated by stage of the software development life cycle.

Since the review results are mainly technical and frequently positivistic, further practice experiences are useful for a critical reflection of the status and future potentials of AI in software engineering.

To provide a validated and critically forward-thinking analysis expert-interviews with software engineers involved at different stages of the software development life cycle and in different fields of engineering [45] are conducted to accomplish the review results. To avoid a

Table 1 Characteristics and levels of education of the interview participants

Characteristics	Cohort (***n*** = 5)
Age in years	29 to 37
Sex	4 male, 1 female
Average years of work experience	9,2
Average number of software companies worked for	3,6
Average number of software products developed	18.5
Level of Education	
Bachelors	1
Masters	3
Doctorate	1

bias towards the potentials of AI, explicitly software engineers who are well informed on but not directly in AI development but in general software engineering have been recruited [44]. The interviewed experts, four men and one woman, are aged between 29 and 37 and all have studied computer sciences. Their average experience on the job is 9.2 years and the all have worked for several software companies and have developed between 9 and 20 software products (Table 1). Four from five interviewees rank themselves as excellent (2) or experienced (2) experts in AI technologies due to their practical experience on the job. Interviewee five indicates average knowledge in the field.

The interviews are semi-structured and audio-recorded and transcribed verbally (appendix). The interview questions explicitly refer to the stages of the software engineering life cycle but allow experts to detail the relevant fields and the relevance of AI at these stages according to their own impetus and experience. This strategy avoids an interviewer's bias concerning focus and interpretation of AI potentials and risks [46]. The interviews are evaluated in comparison and with regard to perceived opportunities, limitations and future potentials of AI at the stages of the software development life cycle.

The integration of review results and interviews provides a new technically founded but still critical perspective on the potentials, limitations and development requirements of AI in software engineering.

AI in the software engineering life cycle

The review results on the potentials, limitations and development requirements of AI applications in software engineering are presented by software development life cycle stage in the following sections:

AI in software project planning

At the stage of software project planning software developers and clients come together to determine the project objectives and customer requirements [47, 48] Software development scheduling and planning is of crucial importance to ensure the technical effectiveness and economic efficiency of software projects [41].

Search-based software engineering is involved with the optimization of project targets e.g. costs, duration and quality under certain constraints and has originated in the late 1990ies. While early algorithms were based on conventional linear programming, increasingly coefficient interdependencies, non-linearities, several decision layers, dynamic conditions and uncertainties have been included in the models [49]. Growing complexity of decision layers and the reference to previous experiences and documentations in external data-bases justify the label of artificial intelligence which is increasingly assigned to innovative scheduling systems:

Duration and cost of projects are contradicting goals at first sight and human planners struggle to harmonize both objectives. AI tools are useful to support this process [50].

Task assignment in the project planning phase is an issue of conflict for developers and project planners and optimal task, time and budget allocation regularly exceed human planning capacities [51]. Artificial intelligence is useful to support software project management at the stage of task assignment and human resource allocation. Conventional scheduling models face the challenge of a very broad search space, comprising multiple input factors and scenarios and usually have to make simplifying assumptions to deliver reproducible results [52]: AI algorithms based on non-linear and self-optimizing algorithms, like ant colony optimization can solve such problems successfully by iteratively reducing decision complexity [53].

Fenton et al. [54] suggest a Bayesian network algorithm for the simultaneous optimization of cost and quality outcomes. Other than conventional optimization models, Bayesian models are able to integrate large amounts of co-determiners and coefficients and cope with missing and uncertain data and subjective judgements. The Bayesian network integrates several levels of cause and effect interrelationships. Stochastic models are used to predict uncertain future conditions. Dynamic Bayesian networks add a time-dependent variable to the model to modify the coefficients and determiners depending on previous developments. Bayesian models require an exact mathematical pre-formulation of the problem set and thus depend on prior human planning and problem analysis.

More recent AI scheduling models rely on self-optimizing iterative algorithms to avoid this problem: Mahadik [52] and Han et al. [53] use a Max-Min Ant System algorithm in Software project planning to minimize the cost and duration of the project by

assigning tasks adequately. Han et al.'s [53] ant colony optimization algorithm approximates ideal work assignment iteratively and delivers an adaptive time and function plan (PERT and Gantt). Mahadik's [52] ant algorithm provides a plan in the form of a task list and an employee allocation matrix which simultaneously optimizes employee allocation and task scheduling. Stylianou & Andreou [51] compare the optimization results of several algorithms in multi-objective task optimization with cost and duration targets in diverse case studies and find AI based algorithms superior to conventional linear planning. Peischl et al. [55] suggest an expert system which is able to select an ideal planning routine depending on the project characteristics. It refers to a knowledge base of earlier projects and uses a constraint-based reasoning mechanism to select and compute relevant items for project task definition.

Athavale et al. [56] use AI to predict the interactions between human entities and their environment in software project operation in order to realize an ideal assignment of tasks and to maximize team performance. The model considers human personality traits and affective states as well as competencies, learnability and individual interactions to compose performing teams as measured by output quantity and development speed. The routine is adaptive to modifications like team changes, work force failure and illness. Practice performance proofs in a real-life team context are outstanding.

Chicano [50] integrate algorithms for task and function scheduling under cost and duration objectives into an adaptive AI model which relies on an external project archive to select a scheduling algorithm adequate to the problem set. Five multi-objective solver algorithms are compared and tested in order to optimize their application in a multidimensional scenario space.

Summing these results up, to date, AI requires human assistance to select an adequate planning algorithm for the respective problem set. The practical application of AI algorithms for devising and scheduling new projects is yet to be done.

AI at the stage of problem analysis

At the stage of problem analysis in the software development life cycle, the problem set is defined in terms of software tools and development requirements by the software development team [41, 47, 48]. Computers have long been employed for problem analysis and the compilation of big data. Analytical AI systems are more comprehensive in the complexity of statistical approaches and dispose of embedded self-reliant learning algorithms that distinguish patterns based on a series of similar or recurring characteristics to enable new creative solutions AI analytics takes recourse to external data bases to get "informed" and further develop established routines [57].

AI analytics is partly applied in at the stage of problem analysis in software engineering e.g. to predict of project success and risk an essential routine required to assess and select prospective software projects:

Elzamly et al. [58] use discriminant analysis to predict and classify risk factors in software development. Discriminant analysis is based on a weighted linear model that includes interrelationships between the risk factors and moderators. Artificial intelligence can be used to calculate risk weights in a dynamic process and cluster risk factors. Hybrid technology road mapping is an AI based planning device applied to assess the potentials and risks of novel technologies and could successfully be applied in the analysis and selection of software projects.

Zhang et al. [59] integrate qualitative and quantitative data to a target-driven technology-road-mapping decomposition model which relies on an expert knowledge data base and uses semantic and fuzzy set analysis strategies in order to evaluate project perspectives based on multiple interdependent determiners.

In future, AI systems could be developed further to decompose complex real-world problems into their fuzzy elements and probabilistic components to structure program codes managing these routines. Software engineering problems can be reformulated as optimization problems to enable computerized solutions. So far however, the structuring of the problem set has to be done by humans, while machines can only reproduce predefined structures and apply probabilistic routines to assess uncertainties [60]. Further development in the field of software problem analysis is required to develop analytic competencies in machines.

AI at the stage of software design

In the design phase, the software project is clearly structured and development tasks are assigned [41, 47, 48]. According to Karpathy [7], software 2.0, used synonymously to neural networks and artificial intelligence in his blog, will develop own program codes, based on a simple input (the problem set). The code will gain in complexity in the process of neural network processing and will not have to be understood or reviewed by human beings any more. Visual recognition, speech recognition, speech synthesis, self-driving car routines and gaming are early manifestations of self-improving and developing program codes.

AI search takes a supportive function in the design of computer games for instance and is applied to model, generate or evaluate content and agent behavior in the game story. AI simulates play throughs and thus contributes to enhance and develop the game, create believable actors and a conclusive computational narrative [61]. AI

software is further employed to continuously test new game routines for practicability and has been employed to train future developers in a university environment [62].

When developers are confronted with probabilistic problem sets and have to develop codes for ill-defined, noisy and incomplete information environments they rely on the stochastic approximations and iterations AI provides [60].

Although neural networks are self-enhancing, they still operate on man-defined routines in the phase of software design. There are tools for specifying and structuring particular problem sets, but the work strategy and the actual design of the software still has to be defined by the human engineer [63]. Although computer aided software engineering, which supports the process of software design by automated activities is common practice today, artificial intelligence implementation still requires clearly structured tasks and the support of human developers to get established. Steps that can be run automatedly have to be defined and integrated in an automated development environment package, which can then perform these functions self-reliantly [64].

Artificial intelligence could in future enhance computer-aided software engineering competencies by intellectual skills and might substitute human activity in that process to some extent [64]. Future AI systems could, according to Lake et al. [63], build causal models that self-reliantly explain real-world phenomena instead of recognizing pre-programmed patterns only. They should be self-adjusting and self-learning in-stead of just optimizing pre-defined routines. The interaction of AI and SE could unleash new creativity potentials in humans by automating routine tasks [65].

AI at the stage of software implementation

Software implementation comprises the actual coding process of the software application [41, 47, 48].

Neural networks have been developed to assist software coding: Processing natural language into software code is a capability of which has been researched since the 1980ies and with increasing complexity of pattern recognition routines has advanced to class-model builders in recent years [66]. Gathered data are transformed into contingent vectors and are used for model training to interconnect code levels systematically [21]. AI software generates prototypes of codes from human language, which then are refined and adjusted by human programmers [66]. AI Classification strategies are useful to directly transform human language and real-world phenomena into pieces of code and software models [67]. Husain et al. [68] develop an AI for automated semantic code search, i.e. the retrieval of relevant code from a natural language query. The software contains several million functions for automatic queries form natural language , which dissect and systematizes human language elements.

Recently, previously established linear statistical models (which are no artificial intelligence), which make part of neural networks, are partly replaced by autoencoding mechanisms, which apply non-linear routines [21]. Autoencoders use a feature fusion process which is trained to use certain elements in order to realize desired characteristics of a data set, e.g. reduce dimensionality, reduce redundant information or select and classify certain classes or categories. Autoencoders consist of several (at least two) layers. The hidden layers inform and train the consecutive ones based on an underlying objective function. The selection of items or other tasks are performed by assigning each selected items weights and bias probabilities, to determine the relevance of the item to the learning process [21]. The so trained neural network can ideally support software engineering by providing comprehensive code elements for certain problem sets. Deep learning strategies thus facilitate and speed up Software development processes [69]. In software coding, deep learning and autoencoding take over computational search and optimization, probabilistic reasoning functions as well as classification routines, coding and prediction functions [60]. Autoencoders can reduce feature sets by fusing several codes [21]. The programming language Swift uses iterative optimization processes which are mathematically implemented by analyzing and optimizing the incremental change of an existing function, to the desired result. In this way, a gradual approximation to defined targets is realized in an automated mathematical process [70].

Today, AI in software implementation however still requires specific and well-defined problem sets e.g. equations to fit and probabilistic environments for simulation. Open analysis in the intention to discover new ideas, new parameters to be optimized or even new problems remains a field of creative human mental activity so far [60].

In future, artificial intelligence could be developed to produce more coherent codes and possibly even implement the code into existing routines self-reliantly. Feldt et al. [12], however, see the risk that software generated automatically could not be understood by human beings anymore and could damage existing routines. AI could ignore risks involved with the automated implementation of autogenerated software codes. Mechanisms to control automatic programming routines will have to be developed to avoid AI related coding risks [12].

AI at the stage of software testing and integration

In the testing phase, the developer and client test the functionality of the software product in practice, identify

and analyze errors and tailor the product to practice requirements [41, 47, 48]. AI uses strategies of pattern recognition and machine learning to support software testing and integration [71]:

Automated software testing refers to the transference of a certain sections of a program into a script, which is then repeatedly executed by a machine, which then collects, stores and interprets the test results [72]. AI Program Browsers check existing codes for necessary changes automatically and suggest changes to the program code in order to make it work [67]. Probabilistic routines support error detection by predicting the likelihood of failure occurrence based on experiences with large data sets. According to Perreault et al. [73], AI based software defect analysis (neural networks) a outperforms classical testing routines.

Software integration refers to the compilation of different codes into a uniform software system [74]. SOA aim at integrating open software standards into firm specific solutions. Rodriguez et al. [75] refer to service-oriented architectures (SOA) to explain how AI can support this process. AI assists developers in integrating different platforms into service-oriented designs and enhances the management of generic quality attributes. AI captures conversation semantics prevalent in different web -based architectures and identifies unifying elements by pattern recognition. AI discovers similar architectures and eliminates redundant code units in the SOA and thus supports developers in clearing up software interfaces so that a contingent SOA tailored to the requirements of specific businesses results [75]. Fuzzing is an automated software testing technique that is not itself based on AI but sometimes combined with AI elements. Fuzzing uses invalid, partly incomplete or random data as inputs to test programs systematically and evaluates the effects, such exceptions to routines take on the program course. The fuzzing results are summarized in the form of an output protocol [76]. Xie et al. [78] and Liang et al. [79] use deep neural networks (DNN) to combine several error routines in order to identify complex code defects. The DNN adapts to program reactions in a metamorphic way in order to identify rare and linked errors and systematically enhance code quality. AI fuzzers prove superior to manual or hybrid fuzzing routines.

Although automated AI based testing and integration functions today are self-improving and use dynamically changing routines, to date human coders are required to define the testing process and requirements to the program, while the test implementation can be done by the machine. A survey among 328 experts comes to the conclusion that about 35% assume that a complete substitution of human programmers by machines in the testing phase will never be possible [79]. AI however abbreviates the testing process and saves manpower to perform, document and evaluate the tests. Time to market and development costs are they reduced [80]. Human control and intervention to date remains necessary to prevent erroneous testing routines and to critically reflect the validity and reliability of test results [72].

AI in software maintenance

In the maintenance phase the software company assists the customer in product application, provides regular upgrade and makes further adjustments upon client requirement [41, 47, 48].

AI instruments successfully support the maintenance and updating process of software to changing requirements in an internet environment [82].

In the maintenance phase, AI can support the classification of user queries, which is useful to classify and direct software users depending on their query pattern (Suresh et al. [82].

Filieri et al. [81] develop a runtime decision engine to adapt an application to respond to unpredictable events. The self-adaptive system continuously reconfigures software components depending on network requirements. This routine saves human support and updating activity and ensures on-time adaptivity, security and program stability.

Using principles of pattern recognition and machine learning, AI equally supports software modernization. In ancient codes, structural information is frequently lost due to poor documentation. AI pattern recognition techniques are useful to extract coherent sets of code. Machine learning functions are used to trace and check their functionality [83]. Pattern tracing functions extract redundant elements from codes and automatically generate implementation artifacts and test software functions [84].

AI neural networks that are trained by deep learning algorithms are useful for software security assessment. AI identifies and simulates attack patterns to discover security gaps, defects and errors in a targeted way [69]. Neural network error and security gap tracking works by slicing software code into formal routines prone to typical attack patterns in a systematic way and exploring a broad set of viral strategies to each element. Neural security assessment networks reach an accuracy of more than 90% in an empirical test on a network architecture [85].

In future, AI software systems could be useful to manage critical large-scale software infrastructure, like servers, and adapt these to environmental changes or new unexpected conditions. To date, however, there is no single AI system that could manage this task self-reliantly [86] (Davis et al., 2016). Lacking human understanding of autonomously regulating AI units could induce self-enforcing cycles which would be beyond

human control risks. Unmanaged AI autonomy could entail unpredictable risks to electronic and even physical infrastructures [12].

Empirical results

From the review of previous studies in the field of AI in software engineering it is obvious that there is significant uncertainty on the remaining potentials and risks of AI. AI comprises several novel technologies and their development lines are still open. To accomplish and validate the review results on potentials, risks and future perspectives of AI in software engineering, interviews with five software developers according to the scheme described in chapter 3 have been conducted and are evaluated comparatively in the following.

Opportunities of AI for software development

The participants agree with earlier studies that as of today AI tools have facilitated the software development life cycle. In correspondence with the review, the interview results on opportunities of AI are evaluated by life cycle stage.

Project planning

As of today, AI indirectly enhances project planning mechanisms, according to participant 2. The analysis of data pools of earlier projects provides realistic estimates of failure quotas and iteration routines in earlier projects and locates potential areas of difficulties. New software projects can be planned more diligently and customer time and cost expectations are met more reliably on the basis of big data analytics.

Participant 4 explains that AI already has eased the prediction of development timelines and enhances estimates of the necessary development steps, which will provide customers as well as engineers with higher clarity on project cost, timing and outcome in future.

Problem analysis

Participant 3 esteems predictive analysis equipment which as of today accesses large online data pools to predict trends and outcomes of new applications. Predictive analyses enable software designers to plan their products more proactively and adjust to new technological trends in their emergence. Big data analytics has improved the competitiveness of his company in an increasingly dynamic software market.

Software design

AI, according to participant 1, provides structured access to immense amounts of data which are retrieved from earlier similar projects, for instance. The number of expected bugs and their location is reliably predicted on that basis and error avoidance routines are established more effectively. AI has sped up the design speed of software projects, according to participant 2, by enabling programs to execute routine tasks, which previously had to be done by human developers. These use their freed creative resources for software design to a larger extent now.

The automation of coding and testing routines using AI has accelerated ideation and planning processes, participant 2 and 3 explain. AI thus has got indirect positive effects on the creative processes of software definition (participant 2). Software designers' creative capacities are strengthened since AI takes over pain-taking daily routines (participant 3).

Software implementation and debugging

Participant 1 is working with AI based programming assistants for Kite for Python which automatically document software codes and provide debugging routines that offer automated suggestions for improvement or deliver code examples for particular problem sets. The tool has reduced software development times and improves output quality. Critical developer resources are freed for more creative jobs than debugging and routine memorization.

Participant 2 estimates the high quality of AI for bug detection and the prediction of future test outcomes. AI debugging instruments discover links and integrations across data automatically, which eases the identification of anomalies and possible inefficiencies in codes (Participant 4). The automation of software testing routines saves development costs and facilitates the job of software engineers (Participant 2).

Participant 3 is experienced with automated code compliers, which support the transformation of high-level programming language codes in machine-executable instructions. AI guided compliers do this job much faster than former manually directed compilers, which enhances software development efficiency.

According to participant 5, AI improves the efficiency of software delivery processes, it eases team collaboration and the integration of customer feedback in code.

Software testing & integration

Deep learning and machine learning today enable systems to integrate apps more comprehensively and thus support limited human reflection capacity (Participant 1). Instead of reading of lengthy documentations and debugging codes, software developers today have gained more freedom due to AI support. Participant 4 sees AI-based testing and compilation software as a personal assistant, providing him with the required hints and information. Participant 5 agrees that AI greatly speeds up the process of debugging according to his practice experience. AI tools trace bugs through the code and

enable their systematic elimination. At the same time, they predict errors early in the programming stage and in this way enable programmers to prevent bugs in their codes from the beginning. This technology is effective as of today and will certainly further evolve in future.

Software maintenance

AI maintenance instruments could in future support businesses in clearing their software products of redundant features. AI tools today identify unnecessary and redundant (double) routines and processes in the course of automated run throughs. AI is more reliable and follows a more complex analytical approach than human analyzers possibly can. While these tasks will be fully automatized in future, human engineers will gain creative potential for planning and design tasks (participant 3)

Limitations of AI in software development

Participant 4 explains that software developers will keep a leading and defining role in the development and improvement of software, since creative not just rationally thinking minds are required to innovate in software solutions. AI in his opinion can never be creative. Participant 5 agrees that developers can foresee and advocate for change, while AI routines can only apply and process existing knowledge. Participant 5 admits that the potential to draw on infinite data bases of knowledge is the most fascinating promise of AI, however explains that predicting the future of AI based on its present development is impossible.

Certainly, software developers will be required to use their creative potential to an even larger extent than before in order to avoid their substitution by machines for routine jobs. Software developers have to keep innovating, improving and learning to use AI effectively in their daily practice (Participant 2) and as participant 3 asserts – certainly will have to be "smarter" in future in order to use novel AI technologies to their potentials. Another final limitation of AI-based software engineering is that most tools are not available yet. We cannot evaluate tools that might be available at some point in the future with respect to their practical relevance.

Further development requirements to enhance AI applicability in software engineering practice

The participants classify future potentials of AI and expected role of human software developers in that context:

Participant 1 expects that AI will manage huge data volumes even more effectively in the near future and provide critical information to project structuring and planning. Problems that become obvious in the process of software development only today, will be avoided from the beginning in future. The development of AI tools to more effectively access and structure information stored in big data pools is preconditional to this development (participant 2).

Software developers have got the challenging role of utilizing AI technologies in engineering practice (participant 5). According to participant 1, the role of software engineers is to expand their understanding for the "disruptive role of AI" [Disruptive technologies are innovations, which fully substitute the success or market advantage of an existing product and turn the investments of other market participants obsolete.]. Software developers should understand themselves as creative innovators and leading parties in change processes. As such they should welcome AI innovations in software design and development (participant 5) and should promote enhanced AI application and integration. The application of AI instruments in software engineering meets the market requirement for self-adapting and self-learning software products (participant 2).

AI can, according to participant 3 make software development processes "faster, smarter and more efficient". The future potential of AI is high considering the fast and eminent progress of software development in other fields (participant 3). Participant 5 sees particular potentials of AI for rapid prototyping, which could benefit from big data analysis and neural networks. AI will further facilitate the coding process by automatic routines allowing to put language into code and or by recognizing visual objects.

Although according to participant 1, theories governing software development will remain the same in the future, AI has got the potential to speed up software development and enhance development efficiency. AI programs could take recourse to knowledge resources which due to their complexity and size are inaccessible to the human mind and this knowledge will "upgrade developers' competencies (participant 1). According to participant 5, the largest potential of AI lies in the utilization of big data structures in the programming process. The availability of a coding data base will enable software developers to use their creative and innovative potential to a larger extent and to the benefit of the final software product.

Discussion

Summative integration of interview and review results

Table 2 summarizes technologies, achievements, limitations and future development potentials of AI for the six stages of the software engineering life cycle as available from previous studies and the interviews. The review results are indicated by bullets and additional insights gained from the interviews are indicated by checks.

The review has shown that the basic principles and technologies underlying AI supported software engineering are

Table 2 Overview on review results by software life cycle stage

Life Cycle Stage	Technologies	Achievements of AI	Limitations	Perspectives
Project planning	• Search-based software engineering • Probabilistic planning • Ant colony optimization • Bayesian network algorithm	• Cost & duration prognosis and optimization • Effective task assignment • Efficient delimitation of search and scheduling space • Improvement of quality outcomes • Improved project planning • Usage of data pools of pervious experiences • Time and cost targets are met	• Manual definition of adequate algorithm • No creative potential of AI	• Selection of ideal planning algorithm • unleash creative potential in human developers • Rapid AI based prototyping
Problem analysis	• Self-learning algorithms • Big data strategies	• Success and risk prediction of software • Evaluation of expert knowledge information pools • Predict trends and programming outcomes	• Causal problem analysis is done by man, while machines only assist	• Decomposition of complex problem sets for systematic analysis and optimization
Software design	• Search based software engineering • Probabilistic planning	• Analysis of conclusiveness of code or story (in gaming) • Test program logics • Probabilistic analysis ✓ Structured access to previous design patterns ✓ Free creative potential and ideation process by taking over routine tasks	• Basic structure is man-made and only checked by machine • Automated routines have to be clearly defined ✓ Higher technical requirements to developers	• Human like skills to interpret real-world phenomena self-reliantly by own learning • Application as a comprehensive data base
Software implementation	• Artificial neural networks/ deep learning • Non-linear statistical analysis • Probabilistic routines	• Natural language processing into code • Autoencoding routines • Automatic debugging and improvement routines • Reduced implementation times & costs • Improved team collaboration	• Dependence on well-defined problem sets and man-prepared structures	• Self-reliant coding and routine implementation • Loss of human control • Big data as reference
Software testing & integration	• Big data • Pattern recognition • Machine learning	• Checking and testing of scripts • Probabilistic error prediction using big data • Abbreviation and cost efficiency of test process • Integration of existing programs (SOA) • Efficiency gains by automated debugging & compiling	• predefinition of control routines is required • Smarter developers are needed to handle automated routines	• Higher self-reliance of testing and integration • Software developers as innovation protagonists
Software maintenance	• Pattern recognition • Artificial neural networks	• Classification of queries; evaluation of errors • Self-adaptive software routines • Clear redundant code • Speed up and ease maintenance	• Man-defined task sets and structures are required • Human control of results	• Higher self-reliance and independence of maintenance and repair functions

similar across the life cycle stages. Artificial intelligence has proven for the automation of routines and analytical processes, when the fundamental structure and organization of the process is clear and its implementation requires lengthy calculation or the analysis of huge data volumes. However, AI comes to its limits when novel insights are sought and new problem sets are meant to be discovered and, innovative routines have to be developed. These fundamental activities so far remain at the hands of human designers and developers. Future AI routines could become more self-reliant if they could compose new tasks and solutions without human support. However, this development would

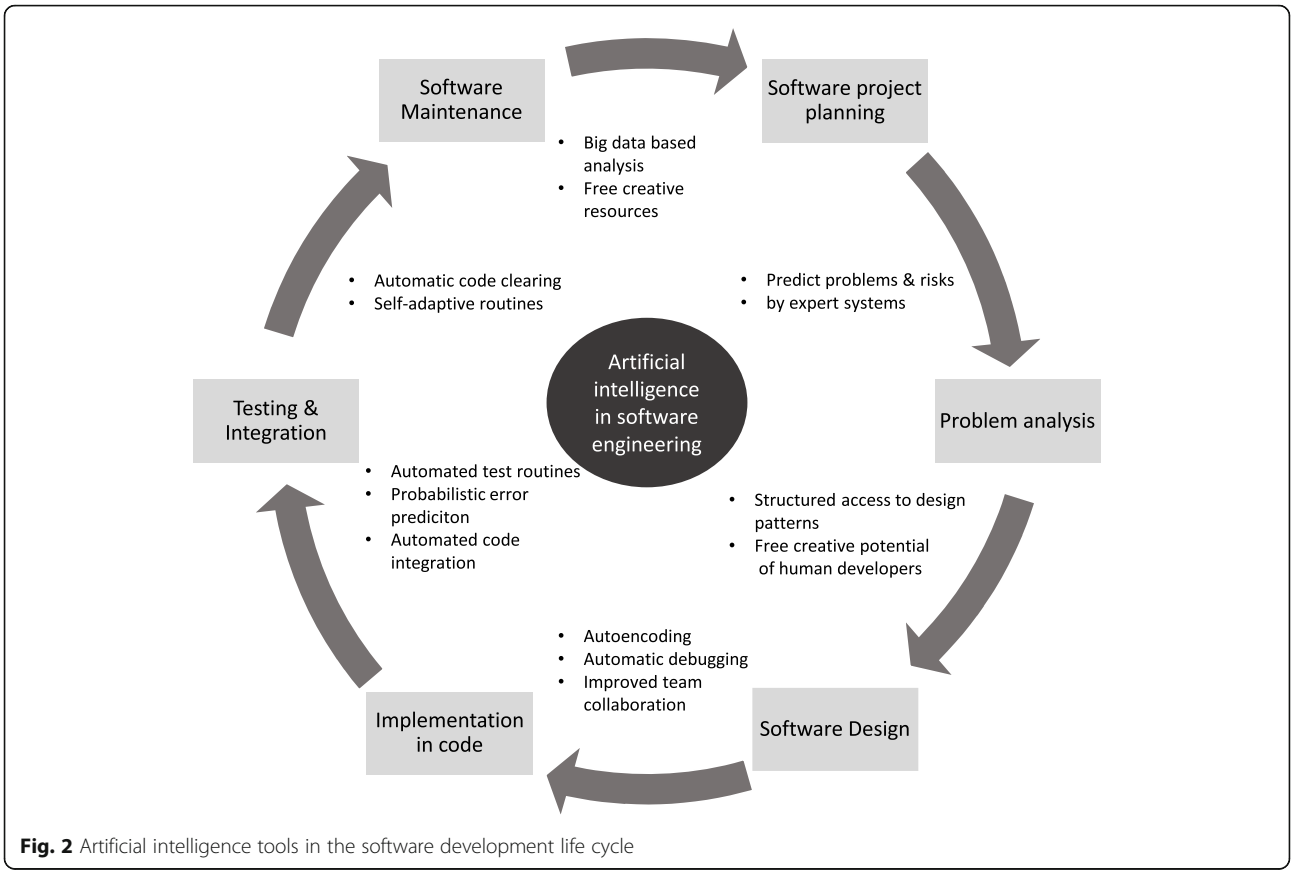

Fig. 2 Artificial intelligence tools in the software development life cycle

entail the initially mentioned risks that man would lose understanding and control of machine-made routines and electronic systems.

Comparing the review and the interview results, the latter have specifically highlighted the capacity and future potential of AI to support software developers which results in important the economic opportunities to software development companies:

AI tools support human developers by taking over routine tasks at every stage of the software development life cycle:

- At the planning stage, AI supports data base search to retrieve and structure information required for planning processes,
- At the stage of problem analysis, AI is useful to assess risk factors of software development process.
- At the stage of software design, AI searches and structures previously developed similar codes and code snippets.
- At the stage of software implementation AI tools transform human language into code and code into machine language automatically.
- At the stage of testing and implementation, AI provides automatic debugging and error tracing routines and supports the integration of individual software routines into comprehensive architectures.
- At the stage of software maintenance, AI is useful for clearing and adapting old code to new requirements.

The major potential of AI at the present stage thus is the support of software engineering by taking over automated routines, while human developers gain time to focus on the creative aspect of software engineering i.e. the planning and design of new software concepts. This work sharing between artificial and human intelligence contributes to reduce development time, enhances software quality output and thus increases the efficiency and market success of software development companies.

The interview results thus accomplish the theoretical understanding of the contribution of AI to software engineering in the software development life cycle. The application of AI instruments at every stage of the development process results in an efficiency increase of the whole process flow, as illustrated in Fig. 2:

Classification of empirical results in in the context of previous research

The review and particularly the empirical section of the analysis have thus accomplished the understanding of the relevance of AI to the software development life

cycle. By structuring the review of previous empirical studies of AI in software development according to the six stages of the software development life cycle, the study has systematically explored, how AI supports every life cycle stage by technical solutions. The so far limitations of AI in software engineering have been discussed. As compared to earlier reviews in the same field the analysis distinguishes by differentiating six life cycle stage and elaborating on the concrete technologies relevant at each stage. So far technology-oriented studies however lack an analysis of the contribution of AI to the whole software development process, since each team of authors' focusses in particular algorithms and routines.

The interview section of this study has contributed to classify the technological advancements in the context of AI in the software development life cycle. Interviews with practitioners in the application of AI equipment in program design and coding have illustrated how AI tools save human developers' efforts, development time and cost at every stage and thus enhance the creative potential of development companies as a whole.

Earlier studies are partly blurred on the limitations and future development requirements of AI, since they lack this comprehensive perspective. This study has worked out that at present AI in software development takes a supportive function by automatizing routines based on big data analytics and lengthy calculations. AI to date is not creative in the sense that new plans for software projects are suggested, new problem sets are defined or new product ideas are suggested. These functions remain at the hands of human software engineers. But these benefit of the automation and calculation speed of AI tools and gain time and mental resources to focus on creative development tasks.

The interviewees see the future development potentials of AI in software engineering rather in the perfection of the supportive functions of AI than in its evolution towards own innovative and creative capabilities. In this regard the interviewees distinguish from some of the reviewed studies, which partly express the hope that AI will in future be able to decompose complex problem sets and develop humanlike skills of real-world interpretation and self-reliant learning [57, 59, 64]. The divergence in the assessment of AI potentials between technology focused empirical AI researchers and the interviewed software developers, probably is a matter of perspectives: AI researchers dispose of a deep structural understanding of the theoretical capabilities of neural networks. Software engineers are concerned with involved with the daily problems and routines of coding and the aptitude of running AI systems successfully to solve their daily problems. To date, there is a divergence between the theoretical understanding of what AI could possibly achieve and the practical capabilities of available AI programs.

Practical development lines for AI development in software engineering

The interview results provide important information for software developers and development companies on the relevance of AI and strategies of AI integration in the business process, which are summarized in some bullet points for management purposes here.

- AI is a future technology and can support every stage of the software development life cycle by automatizing data research, calculations, debugging, compiling and software integration today already.
- Software developing companies benefit of the application of AI tools which speed up development processes by automation, improve team collaboration by enhancing knowledge documentation and interchange, save developers' resources and time efforts and hence reduce development costs while product quality improves.
- The most important aspect is the potential of AI to free the creative and innovative capacity of human software developers by taking over routine functions. Neural networks have even proven creative in practice tests, e.g. in abstract landscape painting [87]. However, the combination of reason and creativity and the combination of diverse modalities and abstraction levels still make the human software engineer much better than any machine today. Development companies adopting AI tools enjoy a competitive advantage by increasing the innovation potential of their workforce.

Conclusions

Software developers and their companies however have to fulfil some requirements to use the opportunities of AI: Software developers have to continuously adapt their competencies and qualifications to keep up with the dynamic and rapid development of AI tools in software engineering. They have to be open minded to apply these instruments in their daily work practice. Software companies should encourage employees' engagement by providing leadership support and making the necessary investments in new AI programs and hardware infrastructure.

Software developers who ignore the potentials of AI in the software development life cycle and stick to routine jobs, which are more reliably and cost-efficiently done by automated routines risk being substituted and losing their established jobs in the long run. In future, software developers will require higher creative potential and have to be smarter to compete with artificial intelligence. Software development businesses rejecting the adoption of AI risk being pushed out of the market by more innovative competitors who realize software products

faster and at higher quality by relying on AI support. AI is an emerging future technology in software engineering and early adopters multiply their competitiveness.

Limitations and call for further research

Although the combination of systematic review and practitioner interviews in this study has developed some visionary insights on the potentials of AI, which are relevant to developers and mangers alike, the study results remain explorative and require further empirical foundation: The topic AI in software engineering is too broad to be discussed comprehensively within the framework of a journal article and the range of about 60 studies evaluated for the review is not sufficient to analyze all relevant AI technologies, their potentials and limitations comprehensively. Future studies should focus on select stages of the value cycle in order to deepen and validate the qualitative results of this review.

The interview section comprises five in-depth interviews with software developers, which of course is not a representative number. The selection of the interview participants necessarily is arbitrary to some extent and the participants have not been informed on all potential AI technologies since they are software developers not AI researchers in their daily practice. Future research could amend on this problem by bringing AI researchers and software developers together in discussion rounds and assess to what extent developers' requirements are already met or can be supported by AI technology in future. Such an approach would at the same time forward AI research in the field of software engineering. More extensive collaboration between both research fields is desirable and necessary to make AI a comprehensive technology for software engineering in future.

Supplementary information

Supplementary information accompanies this paper at https://doi.org/10.1186/s42467-020-00005-4.

Additional file 1.

Abbreviations

AI: Artificial Intelligence; SE: Software Engineering; SDLC: Software Development Life Cycle; SOA: Service oriented architecture

Acknowledgments

Not applicable.

Authors' contributions

MB was responsible for the interviews, the literature review and the major contributor for the manuscript. JR was responsible double-checking the literature review. OT was a major contributor for the manuscript. All authors read and approved the final manuscript.

Funding

n/a

Availability of data and materials

All data generated or analyzed during this study are included in this published article and its supplementary information files.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]LMIS AG, Osnabrück, Germany. [2]Universität Osnabrück, DFKI, Osnabrück, Germany.

Received: 2 April 2020 Accepted: 6 July 2020
Published online: 26 July 2020

References

1. Makridakis S. The forthcoming artificial intelligence (AI) revolution: its impact on society and firms. Futures. 2017;90:46–60.
2. Acemoglu D, Restrepo P. Artificial intelligence, automation and work (no. w24196): National Bureau of Economic Research; 2018.
3. Friedrich O, Racine E, Steinert S, Pömsl J, Jox RJ. An analysis of the impact of brain-computer interfaces on autonomy. Neuroethics. 2018:1–13.
4. Dhar V. The future of artificial intelligence. Big Data Vol. 4 , N. 1; 2016.
5. Helbing D, Frey BS, Gigerenzer G, Hafen E, Hagner M, Hofstetter Y, et al. Will democracy survive big data and artificial intelligence? In: Towards digital enlightenment. Cham: Springer; 2019. p. 73–98.
6. Kietzmann J, Pitt LF. Artificial intelligence and machine learning: what managers need to know. Bus Horiz. 2020;63(2):131–3.
7. Karpathy, A.. Software 2.0, Blog Contribution in Medium Programming of Nov 11, 2017. 2017. Retrieved from: https://medium.com/@karpathy/software-2-0-a64152b37c35, Access on July 1 2020.
8. Fetzer JH. Artificial intelligence: Its scope and limits (Vol. 4): Springer Science & Business Media; 2012.
9. Russell S, Norvig P. Artificial intelligence: a modern approach; 2002.
10. Carroll JB. Human cognitive abilities: a survey of factor-analytic studies. Cambridge: Cambridge University Press; 1993.
11. Süß H-M. Intelligenztheorien. In: Kubinger K, Jäger RS, editors. Stichwörter der Psychologischen Diagnostik. Weinheim: Psychologie Verlags Union; 2003. p. 217–24.
12. Feldt R, de Oliveira Neto FG, Torkar R. Ways of applying artificial intelligence in software engineering. In: 2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE). New York: IEEE; 2018. p. 35–41.
13. Muenchaisri P. Literature reviews on applying artificial intelligence/machine learning to software engineering research problems: preliminary; 2019.
14. Savchenko D, Kasurinen J, Taipale O. Smart tools in software engineering: a systematic mapping study. In: 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia: IEEE; 2019. p. 1509–13.
15. Russom P. Big data analytics. TDWI best practices report, fourth quarter. 2011;19(4):1–34.
16. Kibria MG, Nguyen K, Villardi GP, Zhao O, Ishizu K, Kojima F. Big data analytics, machine learning, and artificial intelligence in next-generation wireless networks. IEEE Access. 2018;6:32328–38.
17. Alpaydin E. Introduction to machine learning. Cambridge: MIT press; 2020.
18. Ripley BD. Pattern recognition and neural networks. Cambridge: Cambridge University Press; 2007.
19. Tang TA, Mhamdi L, McLernon D, Zaidi SAR, Ghogho M. Deep learning approach for network intrusion detection in software defined networking. In: 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM). Morocco: IEEE; 2016. p. 258–63.
20. Niyaz Q, Sun W, Javaid AY. A deep learning based DDoS detection system in software-defined networking (SDN). arXiv preprint. arXiv. 2016;1611:07400.
21. Charte D, Charte F, García S, del Jesus MJ, Herrera F. A practical tutorial on autoencoders for nonlinear feature fusion: taxonomy, models, software and guidelines. Inf Fusion. 2018;44:78–96.
22. Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P. Natural language processing (almost) from scratch. J Mach Learn Res. 2011;12(Aug):2493–537.
23. Klette R, Koschan A, Schlüns K. Computer vision: Räumliche information aus digitalen Bildern: Springer-Verlag, Berlin; 2013.

24. Shankari KH, Thirumalaiselvi R. A survey on using artificial intelligence techniques in the software development process. Int J Eng Res Appl. 2014; 4(12):24–33.
25. Kakatkar C, Bilgram V, Füller J. Innovation analytics: leveraging artificial intelligence in the innovation process. Business Horizons. 2020;63(2):171–81.
26. Poole DL, Mackworth AK. Artificial intelligence: foundations of computational agents. Cambridge: Cambridge University Press; 2010.
27. Laplante PA. Dictionary of computer science, Engineering and Technology. Florida: CRC Press; 2000.
28. Lu H, Li Y, Chen M, Kim H, Serikawa S. Brain intelligence: go beyond artificial intelligence. Mobile Netw Appl. 2018;23(2):368–75.
29. Bontrager P, Khalifa A, Mendes A, Togelius J. Matching games and algorithms for general video game playing. In: Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference; 2016.
30. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science. 2018;362(6419):1140–4.
31. Bobrow DG, Raphael B. New programming languages for artificial intelligence research. ACM Comput Surv (CSUR). 1974;6(3):153–74.
32. Clark DF, Kandel A. HALO-a fuzzy programming language. Fuzzy Sets Syst. 1991;44(2):199–208.
33. Levinson R. A general programming language for unified planning and control. Artif Intell. 1995;76(1–2):319–75.
34. Wu Y. Object-oriented Programming Course Reform Using Python Language in the Background of Artificial Intelligence. In: 2019 3rd international conference on education, management science and economics (ICEMSE 2019). Singapore: Atlantis Press. p. 2019.
35. Jarrahi MH. Artificial intelligence and the future of work: human-AI symbiosis in organizational decision making. Bus Horiz. 2018;61(4):577–86.
36. Sorte BW, Joshi PP, Jagtap V. Use of artificial intelligence in software development life cycle—a state of the art review, International Journal of Advanced Engineering and Global Technology; 2015. p. 398–403.
37. Padmanaban PH, Sharma YK. Implication of artificial intelligence in software development life cycle: a state of the art review. 2019 IJRRA all rights reserved; 2019.
38. Kisselev, A.. The software Devleopment life Cyle. 2020. Retrieved from: https://www.kisselev.de/, Access on 18 May 2020.
39. Leau YB, Loo WK, Tham WY, Tan SF. Software development life cycle AGILE vs traditional approaches. Int Conf Inf Netw Technol. 2012;37(1):162–7.
40. Ruparelia NB. Software development lifecycle models. ACM SIGSOFT Softw Eng Notes. 2010;35(3):8–13.
41. Seffah A, Gulliksen J, Desmarais MC. Human-centered software engineering-integrating usability in the software development lifecycle (Vol. 8). Luxemburg: Springer Science & Business Media; 2005.
42. Ramadan R, Widyani Y. Game development life cycle guidelines. In: 2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS). Bali: IEEE; 2013. p. 95–100.
43. Velmourougan S, Dhavachelvan P, Baskaran R, Ravikumar B. Software development life cycle model to build software applications with usability. In: 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Noida: IEEE; 2014. p. 271–6.
44. Nusbaum L, Douglas B, Damus K, Paasche-Orlow M, Estrella-Luna N. Communicating risks and benefits in informed consent for research: a qualitative study. Glob Qual Nurs Res. 2017;4 https://doi.org/10.1177/2333393617732017.
45. Vaismoradi M, Jones J, Turunen H, Snelgrove S. Theme development in qualitative content analysis and thematic analysis; 2016.
46. McGrath C, Palmgren PJ, Liljedahl M. Twelve tips for conducting qualitative research interviews. Med Teach. 2019;41(9):1002–6.
47. Ghezzi C, Jazayeri M, Mandrioli D. Fundamentals of software engineering: Prentice Hall PTR; 2002.
48. Mayhew DJ. The usability engineering lifecycle. In: CHI'99 Extended Abstracts on Human Factors in Computing Systems; 1999. p. 147–8. https://doi.org/10.1145/632716.632805.
49. Ferrucci F, Harman M, Sarro F. Search-based software project management. In: Software Project Management in a Changing World. Berlin, Heidelberg: Springer; 2014. p. 373–99.
50. Chicano F, Luna F, Nebro AJ, Alba E. Using multi-objective metaheuristics to solve the software project scheduling problem. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation; 2011. p. 1915–22.
51. Stylianou C, Andreou AS. Investigating the impact of developer productivity, task interdependence type and communication overhead in a multi-objective optimization approach for software project planning. Adv Eng Softw. 2016;98:79–96.
52. Mahadik A. An improved ant Colony optimization algorithm for software project planning and scheduling. Int J Adv Eng Glob Technol. 2014;2(1):6.
53. Han W, Jiang H, Lu T, Zhang X, Li W. An optimized resolution for software project planning with improved max–min ant system algorithm. Int J Multimedia Ubiquitous Eng. 2015;10(6):25–38.
54. Fenton N, Hearty P, Neil M, Radlinski L. Software project and quality modelling using Bayesian networks. In: Artificial intelligence applications for improved software engineering development: New prospects: IGI Global; 2010. p. 1–25.
55. Peischl B, Zanker M, Nica M, Schmid W. Constraint-based recommendation for software project effort estimation. J Emerg Technol Web Intell. 2010;2(4): 282–90 https://doi.org/10.4304/jetwi.2.4.282-290.
56. Athavale S, Balaraman V. Human behavioral modeling for enhanced software Project Management. In: 7th International Conference on Software Engineering; 2013. p. 15–7. sn.
57. Davenport TH. From analytics to artificial intelligence. J Bus Analyt. 2018;1(2): 73–80.
58. Elzamly A, Hussin B, Abu-Naser SS, Doheir M. Classification of software risks with discriminant analysis techniques in software planning development process; 2015.
59. Zhang Y, Robinson DK, Porter AL, Zhu D, Zhang G, Lu J. Technology roadmapping for competitive technical intelligence. Technol Forecast Soc Chang. 2016;110:175–86.
60. Harman M. The role of artificial intelligence in software engineering. In: 2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE): IEEE; 2012. p. 1–6.
61. Yannakakis GN, Togelius J. A panorama of artificial and computational intelligence in games. IEEE Trans Comput Intell AI Games. 2014;7(4):317–35 https://doi.org/10.1007/978-3-319-63519-4.
62. Kalles D. Artificial intelligence meets software engineering in computing education. In: Proceedings of the 9th Hellenic Conference on Artificial Intelligence; 2016. p. 1–5.
63. Lake BM, Ullman TD, Tenenbaum JB, Gershman SJ. Building machines that learn and think like people. Behav Brain Sci. 2017;40:e253.
64. Imam AT, Alnsour AJ, Al-Hroob A. The definition of intelligent computer aided software engineering (I-CASE) tools. J Inf Eng Appl. 2015;5(1):47–56.
65. Jain P. Interaction between software engineering and artificial intelligence-a review. Int J Comp Sci Eng. 2011;3(12):3774.
66. Ammar HH, Abdelmoez W, Hamdi MS. 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; 2012. p. 52.
67. Pawar P. Application of artificial intelligence in software engineering. J Comp Eng. 2016;18(3):46–51.
68. Husain H, Wu HH, Gazit T, Allamanis M, Brockschmidt M. Codesearchnet challenge: Evaluating the state of semantic code search. arXiv preprint arXiv: 1909.0943; 2019.
69. Li X, Jiang H, Ren Z, Li G, Zhang J. Deep learning in software engineering. arXiv preprint. arXiv. 2018;1805:04825.
70. Wei R., Zheng, D., Rasi, M. & Chrzaszcz, B. Differentiable Programming Manifesto. 2020. Retreived from: https://github.com/apple/swift/blob/master/docs/DifferentiableProgramming.md#math-introduction, Access on July 11 2020.
71. Meinke K, Bennaceur A. Machine learning for software engineering: models, methods, and applications. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion): IEEE; 2018. p. 548–9.
72. Mäkelä M. Utilizing artificial intelligence in software testing. Finland; 2019.
73. Perreault L, Berardinelli S, Izurieta C, Sheppard J. Using classifiers for software defect detection. In: 26th International Conference on Software Engineering and Data Engineering; 2017. p. 2–4.
74. Sullivan KJ, Notkin D. Reconciling environment integration and software evolution. ACM Trans Softw Eng Methodol (TOSEM). 1992;1(3):229–68.
75. Rodríguez G, Soria Á, Campo M. Artificial intelligence in service-oriented software design. Eng Appl Artif Intell. 2016;53:86–104.
76. Takanen, A., Demott, J. D., Miller, C., & Kettunen, A.. Fuzzing for software security testing and quality assurance. 2018. Artech House.

77. Xie X, Ma L, Juefei-Xu F, Chen H, Xue M, Li B, et al. Coverage-guided fuzzing for deep neural networks. arXiv preprint arXiv:1809.01266, 3; 2018.
78. Liang H, Jiang L, Ai L, Wei J. Sequence Directed Hybrid Fuzzing. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER): IEEE; 2020. p. 127–37.
79. King TM, Arbon J, Santiago D, Adamo D, Chin W, Shanmugam R. AI for testing today and tomorrow: industry perspectives. In: 2019 IEEE International Conference On Artificial Intelligence Testing (AITest): IEEE; 2019. p. 81–8.
80. Hourani H, Hammad A, Lafi M. The impact of artificial intelligence on software testing. In: 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT). Jordan: IEEE; 2019. p. 565–70.
81. Filieri A, Hoffmann H, Maggio M. Automated multi-objective control for self-adaptive software design. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. Bergamo; 2015. p. 13–24.
82. Suresh SS, Naidu MM, Kiran SA, Tathawade P. Design pattern recommendation system: a methodology, data model and algorithms. ICCTAI'2011; 2011.
83. Alhusain S, Coupland S, John R, Kavanagh M. Towards machine learning based design pattern recognition. In: 2013 13th UK Workshop on Computational Intelligence (UKCI): IEEE; 2013. p. 244–51.
84. Van Hoorn A, Frey S, Goerigk W, Hasselbring W, Knoche H, Köster S, Wittmüss N. DynaMod project. Lübeck: Dynamic analysis for model-driven software modernization; 2011.
85. Adebiyi A, Arreymbi J, Imafidon C. Security assessment of software design using neural network. arXiv preprint. arXiv. 2013;1303:2017.
86. Davis J, Hoffert J, Vanlandingham E. A taxonomy of artificial intelligence approaches for adaptive distributed real-time embedded systems. In: 2016 IEEE International Conference on Electro Information Technology (EIT): IEEE; 2016. p. 0233–8.
87. Art-DCGAN; 2020. Retrieved from: https://github.com/robbiebarrat/art-DCGAN. Access on July 11 2020.

Publisher's Note