Journal of Computer Science

Review

Artificial Intelligence for Software Engineering: An Initial Review on Software Bug Detection and Prediction

[1]Julanar Ahmed Fadhil, [1]Koh Tieng Wei and [2]Kew Si Na

[1]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), 43400 Serdang, Malaysia
[2]Faculty of Social Sciences and Humanities, Universiti Teknologi Malaysia (UTM), 81310 Skudai, Malaysia

Article history
Received: 29-10-2020
Revised: 12-12-2020
Accepted: 16-12-2020

Corresponding Author:
Koh Tieng Wei
Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), Serdang, Malaysia
Email: twkoh@upm.edu.my

Abstract: The need for speed and quality in delivering all software engineering artifacts has inevitably remained the biggest challenge in today's software development environment. While everyone caters to complex software engineering processes, new releases are expected by the market on almost a daily basis. Thus, several Artificial Intelligence (AI) techniques have been introduced that are intensively used in the modern software engineering industry to fulfill market needs. This paper presents the initial results of our review work on software bug detection and prediction studies using AI techniques. Our focus is to (i) identify factors affecting the effectiveness of current software bug detection and prediction techniques and (ii) identify the effectiveness of AI techniques in improving current software bug detection and prediction techniques. The evidence showed that the software engineering domain has utilized artificial intelligence approaches and techniques to facilitate the complex tasks of software bug detection and bug prediction. It mainly demonstrates the significance of merging artificial intelligence with the software engineering domain in terms of reduced overhead and efficient results to enhance the quality of software products.

Keywords: Artificial Intelligence, Software Engineering, Bug Detection and Prediction

Introduction

Conceptually, software engineering mainly concentrates on the scientific and systematic mechanisms involved in developing, maintaining, operating and retiring software products. However, the software development process is considered a very complicated process that currently represents a primarily human activity (Mall, 2018). Software engineering is considered an intensive knowledge activity, which requires extensive application domain knowledge of specific target software. Programming and software development require various categories of previous knowledge in both the programming and problem domains. A final solution should include a combination of both knowledge types to provide efficient and reliable software (Jabangwe *et al*., 2018). Despite the disciplinary and systematic natures of software development, it is subject to various issues and limitations. Simulating the human mind and behaviors is one of the most difficult issues and computer awareness also represents another obstacle for software engineering. Also, sequential approaches and fixed phases are widely adopted process models in software engineering, so, by nature, software products are not flexible (Fitzgerald and Stol, 2017). Furthermore, software that depends on real time becomes difficult to optimize using software engineering. Many costs of software products are attributed to current techniques that are ineffective in managing that knowledge. Therefore, artificial intelligence techniques can be efficient alternative solutions.

Artificial intelligence can be referred to as a fundamental computer science field which mainly focuses on creating intelligent machines and approaches. It includes the design and implementation of various intelligent methods. Artificial intelligence research is extremely technical and deeply specialized. It can be categorized into various sub-categories that are often not communicated effectively (Jackson, 2019). On the other hand, artificial intelligence is also reported as having various types of issues and limitations. However, various traits, including learning, knowledge reasoning and

planning as well as the ability to move, communicate with and manipulate objects, represent fundamental problems in the field of artificial intelligence. General intelligence or strong artificial intelligence is still considered one the long-term goals of the field. Presently, not all features and characteristics of human intelligence have been properly captured. Recently, artificial intelligence research has focused more often on fields of distinguished application, where the full extent of artificial intelligence capabilities is not required (Acemoglu and Restrepo, 2018). Both software engineering and artificial intelligence are considered critical fields of the computer science. The specifications of both software engineering and artificial intelligence have been separately developed during the last decades with little exchange of research results. However, both computer science fields include various features and drawbacks. Therefore, integrating artificial intelligence with software opens the door for various possibilities and ideas.

Software engineering is considered a formal and scientific approach used to implement, develop, evaluate and superannuate software products. Despite the disciplined and systematic approaches used in software engineering, it still has multiple issues and limitations. Initially, the simulation of the human mind or its actions was considered a complex task to be handled with the utilization of software engineering. However, software engineering does not support the computer consciousness and quick solving of NP-complete problems, which are impossible to process in polynomial time (Tunio *et al*., 2018). Finally, the majority of software engineering process models utilize the sequential approach and fixed phases, which result in inflexibility of software. Ultimately, due to the low cost of building software, formal methods for engineering validation are not used widely in software development for real implementation. Software development is considered a profession more than an engineering exercise due to a lack of firmness in the fundamental processes of design improvement and validation (Ramirez *et al*., 2018).

The tremendous growth of research on emerging techniques in artificial intelligence for implementing software engineering in recent years has resulted in an increasing number of publications and projects (Kumari and Kulkarni, 2018). A dedicated number of journals and conferences have focused on artificial intelligence integration with software engineering to propose various researches on this attractive topic. The main goal of proposing artificial intelligence approaches is to minimize the market time and provide software system quality enhancement. Thus, the majority of artificial intelligence approaches still utilized by researchers have had a minor impact on the tools and processes of practicing software engineers (Kulkarni and Padmanabham, 2017).

In this review paper, we attempt to present the latest efforts and ways of utilizing artificial intelligence in software engineering. These efforts can help to bridge the gap between researchers' proposals and artificial intelligence-based applied techniques to handle issues in software engineering.

The rest of this paper is organized as follows. Section 2 presents the background of the software lifecycle and potential factors which need to be considered when integrating software engineering and artificial intelligence. Section 3 presents the methodology used for this paper for reviewing the latest approaches for integrating software engineering and artificial intelligence. Section 4 provides a detailed review of various artificial intelligence approaches used in three main complex trends of software engineering, including cost estimation, bug detection and bug prediction. A discussion then investigates the results of applying various artificial intelligence methods for software engineering.

Background

In this section, a brief introduction to the software development lifecycle is introduced. The IEEE 12207 standard comprises the main framework for the software development lifecycle process. It mainly provides well defined terminology, which can be utilized by all clients. This standard includes all tasks, processes and activates related to the development process of any software system, including standalone software or software as a service. The lifecycle covers requirements for all phases of analysis, design, implementation, testing and evolution, as shown in Fig. 1.

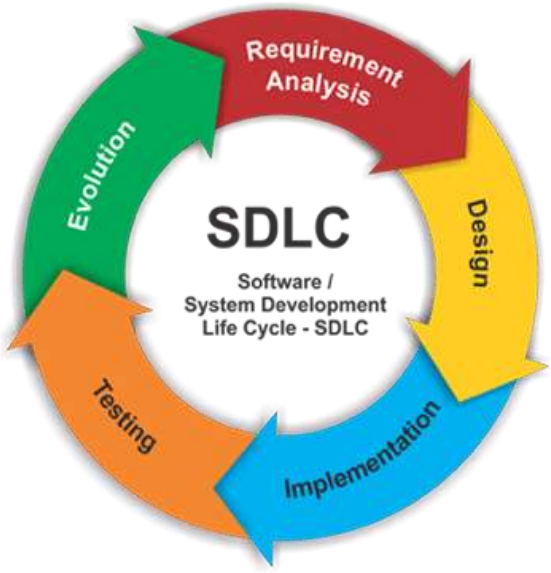

Fig. 1: Lifecycle of software system development

System and software architectures play critical roles in assigning management activities when software systems are being developed and maintained. The activities and associated documents of the development process workflow are provided by the IEEE standard. The activity of the system architecture design that generates the software architecture, the software requirements and allocation description documents represent the highest level of the system architecture and mainly define the hardware and software components of that system. After that, these components are developed separately and concurrently. The development of software components is initialized by the software requirements analysis phase, which results in the documentation of the software requirements specification. The software architecture design phase then generates the software architecture description documents and software interface design description. The software coding and testing activities come in the next stage.

Since the artificial intelligence concept was postulated, an increasing utilization of artificial intelligence mechanisms, algorithms, tools and implementations have been reported in various fields, particularly in the field of software engineering. There is increasing research interest in emerging artificial intelligence techniques in software engineering. However, various factors were considered before the emergence of artificial intelligence and software engineering. These factors include communication levels, objectives, problems, motivations and drawbacks (Shehab *et al*., 2020).

Communication Level

The main goal of software engineering is to study, build and enhance the quality of delivered software. Efficient software involves the reduction of time and cost overheads in the building stage. These goals focus on various issues, including reliability, productivity, software reusability and software maintenance. Artificial intelligence is similar to software engineering in both targets and issues (Wangoo, 2018). A solution for these issues has been the main target of research that has focused on these issues. The main problem in interacting at the solution level is that there is a chance for solution exchange from one domain to another. Thus, proposed research needs to be related to problems rather than solutions, where proposed solutions must fit existing problems. Several misconceptions between artificial intelligence and software engineering are caused by attempts to bridge the gap between these two domains at the solution level without reference to objectives. Software engineering contrasts with artificial intelligence in that it is considered an engineering discipline for software quality, efficient development and maintenance.

Objective Level

The second factor is that artificial intelligence and software engineering have different purposes, objectives and goals. Although both concepts are involved in the software development lifecycle, both have different targets. The main target of artificial intelligence is to develop intelligent applications or devices. On the other hand, software engineering mainly targets the development of reliable and high quality applications within stipulated time periods in accordance with disciplines, approaches and optimized costs (Erlenhov *et al*., 2019).

Problem Level

Interaction between artificial intelligence and software engineering can result in various problems because both concepts are meant to handle different problems. Artificial intelligence handles issues that need solutions regarding methods that are not well understood (Afzal and Torkar, 2008). Therefore, artificial intelligence researchers have proposed multiple powerful tools and methods for programming and associated techniques. Software engineering handles issues related to processes used for the simulation of human behavior and computer consciousness that can be achieved within the limits of the discipline. The sequential approach is the major approach, which is used by process models in software engineering. However, engineers have experienced high levels of difficulty in implementing real-time software utilizing software engineering approaches (Ammar *et al*., 2012).

Motivation Level

Artificial intelligence and software engineering have different implementation approaches. Artificial intelligence handles issues related to the implementation of human behavior and emotion and software intelligence. Software engineering, on the other hand, handles issues related to productivity, maintenance and software reliability (Ammar *et al*., 2012). Despite these differences, there are various motivations which push toward the integration of artificial intelligence methods, techniques and tools within software engineering. These motivations include:

1. The concept of automatic programming is a new paradigm for software engineering in research which can directly result from artificial intelligence software integration
2. The development and maintenance environments of artificial intelligence need to be fit for direct application to the software engineering process
3. Artificial intelligence methodology and approaches need to be applied to the design process of software building

4. The rapid architecture of the artificial intelligence model is useful in the software engineering model

Drawbacks Level

Many reasons offered in suggesting approaches for merging artificial intelligence and software engineering are not applicable (Harman, 2012). The reasons for this pessimism include:

1. The sequential approach and non-flexible phases adopted by a majority of software engineering process models do not fit models proposed by artificial intelligence
2. Human behavior is difficult to simulate utilizing software engineering
3. Software engineering does not support the computer consciousness concept
4. Practitioners in the subfields of artificial intelligence, which are deeply divided, overwhelmingly fail to communicate with each other, making it more difficult to communicate with software engineering practitioners

Artificial Intelligence in Software Engineering

In this study, artificial intelligence models are investigated to provide the required topics programmers and researchers need in order to investigate, understand, communicate and discover the advantages and disadvantages of merging artificial intelligence approaches with software systems. We consider few domains of artificial intelligence including data mining, neural networks, knowledge-based systems, fuzzy logic and machine learning. Recent papers are reviewed to investigate which artificial intelligence domains have been utilized to facilitate software engineering models.

Various software engineering problems utilize data mining as an expected solution. Various researches have been proposed that focus on three problem areas from the perspective of data mining in software engineering. These categories include software error prediction, software cost estimation and bug prediction due to software changes (Wang and Srinivasan, 2017).

The bug detection in the initial stages of the software development process is very important for quality assurance and it has been widely adopted for a long time. Testing and bug fixing in software engineering are very expensive and require high levels of resources.

Many researchers have used artificial intelligence approaches to detect and predict software bugs in the early stages of software development. These approaches include logistic regression, principal component analysis, logical classification, layered neural network models, discriminant analysis and holographic networks (Iqbal *et al*., 2019). The back-propagation learning technique has been used to build neural networks. Performance has also been investigated through predictive validity, misclassification rate, cost verification and achieved quality (Lamba and Mishra, 2019).

Detection of Software Bugs

Software bugs are very critical when the software reaches the production stage as bugs minimize client satisfaction and thus threaten software companies' reputations (Uddin *et al*., 2017). Software bug detection is one of the biggest areas to attract researchers' efforts to reduce the consequences of these bugs. Recent researches have argued that up to half of developers and software testers efforts are spent on avoidable work (Hindle and Onuczko, 2019), the majority of which results from system defects (Wikar, 2019). The process of defect management is shown in Fig. 2, which includes defect prevention, deliverable baseline, defect discovery, defect resolution and process improvement. Thus, defect discovery presents a fundamental stage for later stages in which defects can be handled efficiently.

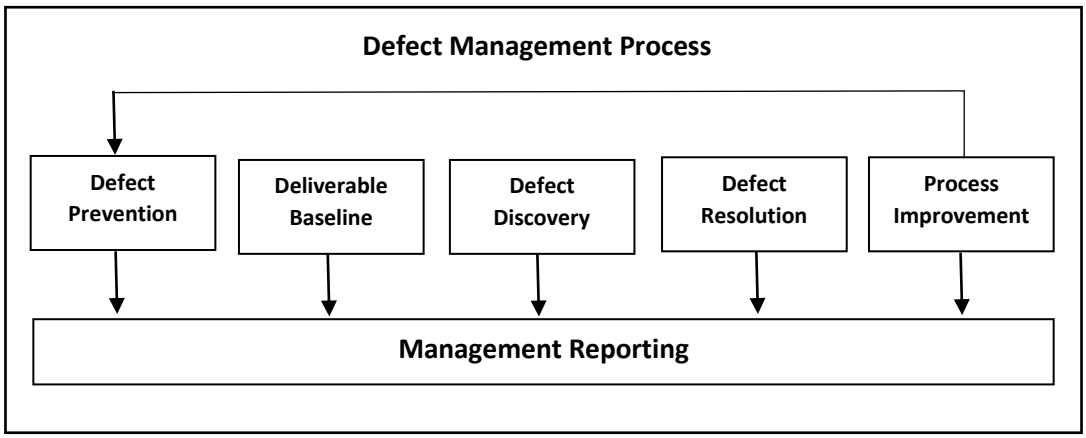

Fig. 2: Defect handling process

Table 1: Software engineering bug detection approaches using AI techniques

Proposed work	Main features and ideas	AI techniques
A mining approach to obtain the software vulnerability characteristics (Li *et al*., 2017b)	It proposed a mining algorithm for vulnerabilities to efficiently detect and analyze the essential software vulnerability characteristics depending on data-mining approaches.	Data mining
NAR-miner for discovering negative association rules from code for bug detection (Bian *et al*., 2018)	The NAR-miner system was designed to extract negative association programming rules automatically from large-scale systems and then detect their violations to find bugs.	Data mining
An ontology-based approach to automate tagging of software artifacts (Alqahtani and Rilling, 2017)	A completely automated classification and tagging approach was proposed. It has the ability to extract security tags from texts with no need to train the system on manual training data.	Ontologies
Software behavior decision trees for defect detection technology (Chen *et al*., 2018)	This layered detection technology is based on a software behavior decision tree model.	Decision tree
Test-driven ontology development in Protégé (Schekotihin *et al*., 2018)	An ontology was proposed based on both logic and test-driven development approaches. It was mainly fed by a queue from test-driven development in the software engineering field, then processed and detected bugs were presented.	Ontology
DWEN: Deep word embedding network for duplicate bug report detection in software repositories (Budhiraja *et al*., 2018)	The proposed approach mainly works on detecting and tagging duplicate bug reports. This approach is important to mitigate same bug assignment to different developers. Detection of bugs and their elimination can significantly reduce the cost of the software development process.	Neural network
MODE: Automated neural network model debugging via state differential analysis and input selection (Ma *et al*., 2018)	This proposed debugging approach mainly works by conducting differential analysis on the model state to detect and analyze the internal model features which cause bugs. It then starts training input selection, which is like program input selection in regression testing.	Neural network
DeBGUer: A Tool for bug prediction and diagnosis (Elmishali *et al*., 2019)	DeBGUer was proposed as a web-based application that can be used to predict and segregate software bugs. It was implemented on different levels for learning, diagnosing and planning (LDP) the approach. The LDP paradigm was a newly introduced approach for integrating artificial intelligence and software engineering for detecting software bugs and for the correction process.	Fault prediction model
An artificial intelligence paradigm for troubleshooting software bugs (Elmishali *et al*., 2018)	It integrated three artificial intelligence technologies: 1. Machine learning, which worked on learning from the structure of the source code and history revisions of failures to identify software components that likely have bugs, 2. Automated diagnosis, which included the software component specifications which require modification to fix detected bugs, 3. Automated planning, which planned extra tests when required to enhance the accuracy of bug diagnostics.	Machine learning

The main idea is that the high cost of software testing challenges practitioners to look for knowledge useful in detecting defects that exist prior to the testing process (Sheneamer and Kalita, 2016). In achieving this, the predictors of software bugs are basically data mining approaches that help in prioritizing the software modules list to be used in tests to effectively assign limitations to testing resources and thus discover most of the existing defects with minimal effort and cost (Pandey *et al*., 2017). There is a positive trend in the number of studies applying intelligent techniques to agile software development (Perkusich *et al*., 2020). The most popular ones are effort estimation, requirements prioritization, resource allocation, requirements selection and requirements management (Xie, 2018). Table 1 shows recently proposed approaches for software bug detection using artificial intelligence approaches.

Prediction of Software Bugs

Software bugs can also result from software configuration changes at run time. This type of bug can result in more catastrophic effects (Zhang *et al*., 2016). Therefore, researchers started investigating the utilization and the integration of artificial intelligence approaches for bug prediction when there is a change in the software configuration or for input which has a high probability of generating bugs. The proposed approaches work on allowing high-risk updates to be directly identified after the code commit process when the changes happen rather than waiting for completing the whole software module and entering the implementing phase (Lanza *et al*., 2016).

The efficiency of defect predicting is shown in Fig. 3 (Arar and Ayan, 2015). The defect predictor is very beneficial for the testing phases, which consume high levels of effort and high costs. It can divide the output modules based on what requires more effort and what requires less testing effort.

When the code updates are committed, the change contexts are still fresh in the mind of the developer, which can make it easier for him to analyze for discovered bugs and connect that with other findings. The prediction models based on bug-inducing changes for specific software applications can be implemented depending on information that describes prior software updates.

Table 2: Software engineering bug prediction approaches using AI techniques

Proposed work	Main features and idea	AI techniques
Severity prediction of software bugs (Otoom *et al*., 2016)	It combined strong classification methods with a proposed feature set to enhance bug severity predicting. The proposed approach utilized a boosting algorithm to enhance performance.	Classification algorithms
Software defect prediction via convolutional neural networks (Li *et al*., 2017a)	A framework called Defect Prediction was proposed utilizing a Convolutional Neural Network (DP-CNN). It leveraged the concept of deep learning to generate effective features based on the programs' Abstract Syntax Trees (ASTs).	Deep learning
Software bug prediction using machine learning approach (Hammouri *et al*., 2018)	This was a machine-learning-based software bug prediction model. Three supervised machine-learning algorithms were used to efficiently predict the expected software faults depending on collected historical data. Utilized classifiers include naive Bayes classifiers, decision trees and artificial neural networks.	Naive Bayes classifiers, decision trees and neural networks
BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques (Pandey *et al*., 2020)	This was a primitive classification approach proposed depending on a framework for bug prediction using Deep Representation and Ensemble Learning (BPDET) approaches for software bug prediction. Ensemble Learning (EL) and Deep Representation (DR) were applied. Conventional software metrics were used for software bug prediction. A Staked Denoising Auto-Encoder (SDA) was utilized to deeply represent the software metrics, which is considered a robust learning feature.	Classification algorithms and deep learning
A comprehensive model for using Bayesian network classifiers for software bug prediction prototypes (Pandey *et al*., 2018)	It adopted two different types of classifier to predict bugs: General Bayesian networks and augmented naive Bayes classifiers. They also compared different Bayesian classifiers and networks in terms of their efficiency in bug prediction.	Bayesian network and classifier
Deep-neural-network-based hybrid approach for software defect prediction using software metrics (Manjula and Florence, 2019)	A hybrid approach was proposed which integrated Deep Neural Networks (DNNs) for classification and genetic algorithms for feature optimization. An enhanced version of genetic algorithm was incorporated, which included a novel technique for fitness function computation and chromosome designing.	Neural network
Software bug prediction system using neural networks (Kumar and Gupta, 2016)	This was a software bug prediction model using gradient descent adoptive learning back-propagation techniques.	Neural network

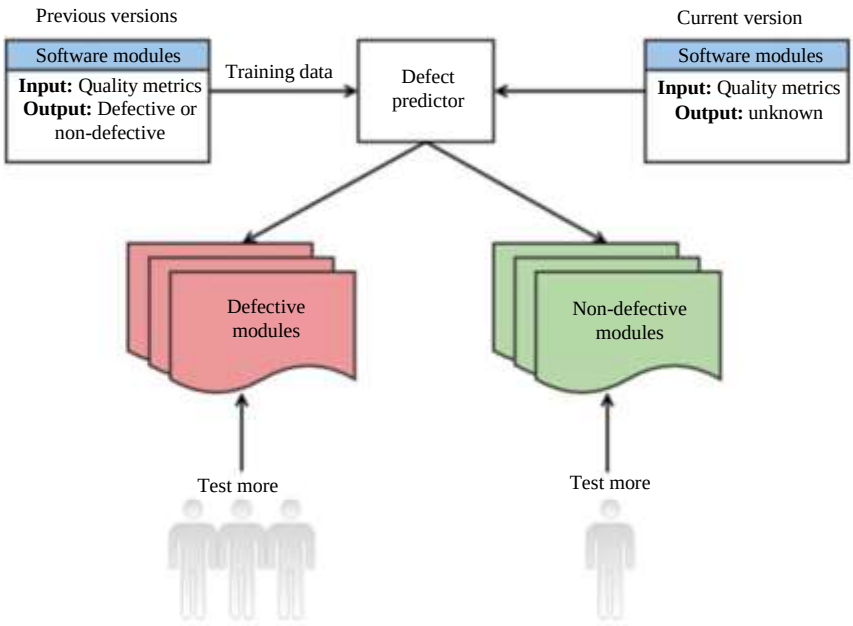

Fig. 3: Defect prediction output

Pervious updates can be fetched using version control systems (Lamba and Mishra, 2019). The automation of software testing and evaluation can be implemented using various AI approaches (Fehlmann and Kranich, 2017). Tools used for implementing such an approach are test stories and test cases that use a formal language to be machine-readable (Fehlmann, 2020).

Table 2 summarizes the recently proposed approaches for software bug prediction using artificial intelligence.

Discussion

The main purpose of this work was to investigate the emergence and integration levels of artificial intelligence techniques in various levels and models of software engineering. An initial review of recently proposed researches utilizing artificial intelligence techniques to facilitate various tasks of software engineering was conducted. This review demonstrates the significance of utilizing artificial intelligence techniques, including machine learning, data mining, neural networks, deep learning and classification methods, to facilitate software engineering tasks. These models succeeded in reducing the overhead of implementation and provided efficient results in applying core concepts of software engineering, including bug detection and bug prediction.

The examples demonstrated reduced effort and costs in terms of the design, implementation and use of artificial intelligence mechanisms to perform various tasks in software engineering. The automation features and learning techniques have significantly enhanced the output of software engineering models and achieved more resilient results in terms of accuracy and overhead reduction.

Neural networks and deep learning have been widely utilized to estimate the required information from current software projects and provide reliable feedback for future updates and projects where artificial intelligence learning has noticeably reduced the required efforts and achieved better results. Various data-mining methods are implemented in software engineering data mining for software bug detection and prediction. Artificial intelligence techniques, including clustering, classification and regression, have changed approaches utilized in visualization modelling and exploratory data analysis. Software developers now more frequently utilize artificial intelligence techniques and business intelligence approaches to support their software skills and facilitate the daily process of decision making for bug detection and prediction.

Conclusion

This paper presents an initial review of the approaches used in the integration of artificial intelligence and software engineering. It investigates how the software engineering domain has utilized artificial intelligence approaches and techniques to facilitate the complex tasks of software bug detection and bug prediction. It mainly demonstrates the significance of merging artificial intelligence with the software engineering domain in terms of reduced overhead and efficient results to enhance the quality of software products. We have also presented an analysis of various artificial intelligence techniques that are utilized in the field of software engineering from different perspectives.

Acknowledgement

This study has been funded by the Ministry of Education (MOE) Malaysia under Fundamental Research Grant (FRGS) project no. 05-01-19-2199FR (5540324). The authors would like to thank the editors and all anonymous reviewers for valuable comments.

Author's Contributions

Julanar Ahmed Fadhil: Collecting, reviewing, synthesizing relevant literature and drafting manuscript contents.

Koh Tieng Wei: Supervising, revising manuscript contents and editing manuscript.

Kew Si Na: Reviewing and editing manuscript.

Ethics

This article is original and contains unpublished material. All authors have read and approved the manuscript and no ethical issues are involved.

References

Acemoglu, D., & Restrepo, P. (2018). Artificial intelligence, automation and work (No. w24196). National Bureau of Economic Research.

Afzal, W., & Torkar, R. (2008, October). A comparative evaluation of using genetic programming for predicting fault count data. In 2008 The Third International Conference on Software Engineering Advances (pp. 407-414). IEEE.

Alqahtani, S. S., & Rilling, J. (2017, November). An ontology-based approach to automate tagging of software artifacts. In 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (pp. 169-174). IEEE.

Ammar, H. H., Abdelmoez, W., & Hamdi, M. S. (2012, February). 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 (p. 52).

Arar, Ö. F., & Ayan, K. (2015). Software defect prediction using cost-sensitive neural network. Applied Soft Computing, 33, 263-277.

Bian, P., Liang, B., Shi, W., Huang, J., & Cai, Y. (2018, October). NAR-miner: discovering negative association rules from code for bug detection. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 411-422).

Budhiraja, A., Dutta, K., Reddy, R., & Shrivastava, M. (2018, May). DWEN: deep word embedding network for duplicate bug report detection in software repositories. In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings (pp. 193-194).

Chen, X. Z., Ding, H. X., Zhang, J., Yang, W. A. N. G., Zhang, G., & Wang, Y. N. (2018). An Artificial Intelligence (AI) Defect Detection Technology Based on Software Behavior Decision Tree. DEStech Transactions on Computer Science and Engineering, (CCNT).

Elmishali, A., Stern, R., & Kalech, M. (2018). An artificial intelligence paradigm for troubleshooting software bugs. Engineering Applications of Artificial Intelligence, 69, 147-156.

Elmishali, A., Stern, R., & Kalech, M. (2019, July). DeBGUer: A Tool for Bug Prediction and Diagnosis. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, pp. 9446-9451).

Erlenhov, L., de Oliveira Neto, F. G., Scandariato, R., & Leitner, P. (2019, May). Current and future bots in software development. In 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE) (pp. 7-11). IEEE.

Fehlmann, T., & Kranich, E. (2017, October). Autonomous real-time software & systems testing. In Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement (pp. 54-63).

Fehlmann, T. M. (2020). Autonomous Real-Time Testing: Testing Artificial Intelligence and Other Complex Systems. Logos Verlag Berlin GmbH.

Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123, 176–189.

Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. International Journal of Advanced Computer Science and Applications, 9(2), 78-83.

Harman, M. (2012, June). The role of artificial intelligence in software engineering. In 2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE) (pp. 1-6). IEEE.

Hindle, A., & Onuczko, C. (2019). Preventing duplicate bug reports by continuously querying bug reports. Empirical Software Engineering, 24(2), 902–936.

Iqbal, A., Aftab, S., Ali, U., Nawaz, Z., Sana, L., Ahmad, M., & Husen, A. (2019). Performance analysis of machine learning techniques on software defect prediction using NASA datasets. Int. J. Adv. Comput. Sci. Appl, 10(5).

Jabangwe, R., Edison, H., & Duc, A. N. (2018). Software engineering process models for mobile app development: A systematic literature review. Journal of Systems and Software, 145, 98–111.

Jackson, P. C. (2019). Introduction to artificial intelligence. Courier Dover Publications.

Kulkarni, R. H., & Padmanabham, P. (2017). Integration of artificial intelligence activities in software development processes and measuring effectiveness of integration. IET Software, 11(1), 18–26.

Kumar, R., & Gupta, D. (2016). Software bug prediction system using neural network. European Journal of Advances in Engineering and Technology, 3(7), 78-84.

Kumari, V., & Kulkarni, S. (2018). Use of artificial intelligence in software development life cycle: Requirements and its model. Int. Res. J. Eng. Technol. (IRJET), 5(8), 398–403.

Lamba, T., & Mishra, A. (2019). Optimal machine learning model for software defect prediction. International Journal of Intelligent Systems and Applications, 10(2), 36.

Lanza, M., Mocci, A., & Ponzanelli, L. (2016). The tragedy of defect prediction, prince of empirical software engineering research. IEEE Software, 33(6), 102–105.

Li, J., He, P., Zhu, J., & Lyu, M. R. (2017a, July). Software defect prediction via convolutional neural network. In 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) (pp. 318-328). IEEE.

Li, X., Chen, J., Lin, Z., Zhang, L., Wang, Z., Zhou, M., & Xie, W. (2017b, August). A mining approach to obtain the software vulnerability characteristics. In 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD) (pp. 296-301). IEEE.

Ma, S., Liu, Y., Lee, W. C., Zhang, X., & Grama, A. (2018, October). MODE: automated neural network model debugging via state differential analysis and input selection. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 175-186).

Mall, R. (2018). Fundamentals of software engineering. PHI Learning Pvt. Ltd..

Manjula, C., & Florence, L. (2019). Deep neural network based hybrid approach for software defect prediction using software metrics. Cluster Computing, 22(4), 9847–9863.

Otoom, A. F., Al-Shdaifat, D., Hammad, M., & Abdallah, E. E. (2016, April). Severity prediction of software bugs. In 2016 7th International Conference on Information and Communication Systems (ICICS) (pp. 92-95). IEEE.

Pandey, N., Sanyal, D. K., Hudait, A., & Sen, A. (2017). Automated classification of software issue reports using machine learning techniques: an empirical study. Innovations in Systems and Software Engineering, 13(4), 279-297.

Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020). BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. Expert Systems with Applications, 144, 113085.

Pandey, S. K., Mishra, R. B., & Triphathi, A. K. (2018). Software bug prediction prototype using Bayesian network classifier: A comprehensive model. Procedia Computer Science, 132, 1412–1421.

Perkusich, M., de Silva, L. C., Costa, A., Ramos, F., Saraiva, R., Freire, A., . . . Gorgônio, K. (2020). Intelligent software engineering in the context of agile software development: A systematic literature review. Information and Software Technology, 119, 106241.

Ramirez, A., Romero, J. R., & Simons, C. L. (2018). A systematic review of interaction in search-based software engineering. IEEE Transactions on Software Engineering, 45(8), 760–781.

Schekotihin, K., Rodler, P., Schmid, W., Horridge, M., & Tudorache, T. (2018, June). Test-Driven Ontology Development in Protégé. In ICBO.

Shehab, M., Abualigah, L., Jarrah, M. I., Alomari, O. A., & Daoud, M. S. (2020). Artificial intelligence in software engineering and inverse. International Journal of Computer Integrated Manufacturing, 1–16.

Sheneamer, A., & Kalita, J. (2016). A survey of software clone detection techniques. International Journal of Computer Applications, 137(10), 1–21.

Tunio, M. Z., Luo, H., Wang, C., Zhao, F., Shao, W., & Pathan, Z. H. (2018). Crowdsourcing software development: Task assignment using PDDL artificial intelligence planning. Journal of Information Processing Systems, 14(1).

Uddin, J., Ghazali, R., Deris, M. M., Naseem, R., & Shah, H. (2017). A survey on bug prioritization. Artificial Intelligence Review, 47(2), 145–180.

Wang, Z., & Srinivasan, R. S. (2017). A review of artificial intelligence based building energy use prediction: Contrasting the capabilities of single and ensemble prediction models. Renewable and Sustainable Energy Reviews, 75, 796–808.

Wangoo, D. P. (2018, December). Artificial intelligence techniques in software engineering for automated software reuse and design. In 2018 4th International Conference on Computing Communication and Automation (ICCCA) (pp. 1-4). IEEE.

Wikar, M. J. (2019). Intelligent software development tools.

Xie, T. (2018, September). Intelligent software engineering: Synergy between ai and software engineering. In International Symposium on Dependable Software Engineering: Theories, Tools and Applications (pp. 3-7). Springer, Cham.

Zhang, T., Chen, J., Yang, G., Lee, B., & Luo, X. (2016). Towards more accurate severity prediction and fixer recommendation of software bugs. Journal of Systems and Software, 117, 166–184.