International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V5I1P117
Eureka Vision Publication | Volume 5, Issue 1, 172-181, 2024

Original Article

Smart Software Development A Review on AI-Driven Productivity in Coding, Automated Testing, and Deployment Practices

Sri Harsha Panchali[1], Usha Mohani kavirayani[2], Krishna Bhardwaj Mylavarapu[3], Jenitha Pilli[4], Prathik Kumar Jannu[5], Javed Ali Mohammad[6]
[1]Information Systems Engineer, CrowdStrike Inc.
[2]Kent State University, MS in Computer Science.
[3]MS in Computer Science, University of Illinois Springfield.
[4]MS in Computer Science, University of Louisiana at Lafayette.
[5]Computer Science Engineering, JNTU Hyderabad.
[6]Masters in Data Science, New England College.

Abstract - A novel approach to software development called "smart software development" makes use of artificial intelligence (AI) to boost output, enhance overall quality, and produce more dependable software across the Software Development Life Cycle (SDLC). It provides a complete overview of how AI-based methods can be used in the areas of coding, automated testing, and deployment practices. Specifically, discuss how machine learning, natural language processing, deep learning, and search-based optimization are used to automate the generation of code, improve the process of reviewing code, support enhanced refactoring using an intelligent algorithm, and enable a more effective detection of defects. In addition, review the use of AI-enabled techniques within the DevOps processes, including smart solution generation, proactive and intelligent monitoring and alerting, self-healing systems, and automated deployment. It synthesis of the most current research demonstrates an increase in the level of productivity gained through AI-enabled techniques, a decrease in the amount of manual work, as well as an increase in the quality of the software produced. The summary shows that for modern software systems to be successful, the integration of all components (from development through testing to deployment) with AI-based techniques is crucial for achieving the highest level of efficiency and adaptability.

Keywords - Smart Software Development, Artificial Intelligence, AI Assisted Coding, Devops, CI/CD.

1. Introduction

Software engineering has been transformed by Advances in Artificial Intelligence (AI) that enable a system to learn from historical data and software development materials. Machine learning (ML), natural language processing (NLP), and search-based techniques are increasingly being used to automatically comprehend the code, identify errors, and facilitate decision-making [1]. In this way, human heuristics for development are reduced and adaptive systems that able to automatic their performance enhancement over time become possible. This is the ground of intelligent and automated software engineering. Over time, Artificial Intelligence (AI) has transitioned the computer industry away from manual methods for creating software products to a more efficient, intelligent and data-centric approach [2]. By utilizing machine learning algorithms and knowledge-based systems, automation has been introduced to code audits, code defect prediction and software test execution thus reducing the labour required to create software and enabling increased productivity within the overall software development process.

Accomplished programmers can now increase their productivity through AI-prompted programming help, including its ability to automate tedious routine programming tasks. For example, techniques such as probabilistic modelling and deep learning enable programmers to learn standard ways of coding [3]. When paired with large code libraries, these techniques can be used for applications like the auto-fill feature, detection and reporting of bugs, and naming of classes, objects, or methods. Intelligent programming tools allow programmers to create consistent and clean code while maintaining the same level of software quality, reduce the amount of cognitive work required to produce their programming code, and increase the speed at which they produce software.

The importance of software testing and the resources needed for it are causing many to consider using automation methods based on AI. Techniques that use machine learning along with search based methods for testing assertively generate test cases, prioritize testing activities, detect faults automatically [4]. The application of intelligent testing techniques allows for more effective testing while reducing the time for executing tests and the effort involved in manually executing the tests. Furthermore, by using the information from past tests and the behaviors exhibited by the system AI (artificial intelligence) can accelerate the speed with which feedback is provided to developers and increase the reliability and ease at which software is developed in continuously developed environments.

The use of AI throughout the software development lifecycle (SDLC) has become a growing trend, demonstrating higher productivity, greater quality, and improved efficiency via the automated use of these technologies to assist in all phases of the SDLC [5]. Using AI methods such as Machine Learning (ML) and Search Based Optimization (SBO) to support the key activities of software development (Code Generation, Defect Detection, Test Automation, and Release Decision Making) results in less development time and a faster delivery of applications without sacrificing reliability in complex and dynamic systems.

The use of AI in software engineering independently has shown to be less impactful than when AI techniques are applied across the entire SDLC (Software Development Life Cycle) to create significant efficiencies [6]. Machine learning models to help with coding decisions, as well as making automated testing decisions, allow for the development of continuous feedback in all stages of the software development lifecycle and continue to promote smarter development workflows while reducing manual input into the process and improving the high-quality delivery of software in a scalable manner.

1.1. Structure of the paper

The structure of the paper, foundational of Smart Software Development in Section II, and increased coding productivity due to AI in Section III, followed by ai in automated software testing in Section IV, AI-assisted deployment and devops practices in Section V, in section VI discuss the literature study and potential future developments in this area in Section VII.

2. Foundations of Smart Software Development

By combining the use of Artificial Intelligence (AI) and Automation with data-based techniques for coding efficiency, Testing Effectiveness and Deployment Reliability, Smart Software Development improves the overall process throughout the Software Development Lifecycle (SDLC) enabling developers to manage, build and deploy Adaptive Software Systems, which are improved, scalable and high-quality systems [7].

2.1. Key AI Technologies in Modern Software Engineering

Neural networks, knowledge-based systems, supervised and unsupervised learning, and reinforcement learning are examples of AI technologies utilized in contemporary software engineering. These technologies allow for automation of development tasks, provide better decision-making capabilities as well as enable intelligent coding, testing and deployment operations.

2.1.1. Machine Learning

Contemporary software engineering outside of research labs is increasingly incorporating machine learning (ML) approaches to not only automate but also facilitate by a wide degree various tasks which are the part of the development lifecycle. A variety of conventional supervised and unsupervised machine learning techniques have been used for defect prediction, code recommendation, effort estimation, and maintenance prioritization. Machine learning models, through the transformation of code and its metadata into features or embeddings, are then capable of making predictions of where defects may occur, recommending changes, or pointing out the most risky areas of code thereby cutting down on the manual work and making the process more stable.

2.1.2. Natural Language Processing

Since numerous software artifacts e.g. requirements documents, bug reports, code comments, and documentation are expressed in natural language, NLP techniques have been incorporated into software engineering processes. NLP is instrumental in parsing, classifying, and getting the useful informational parts from the textual artifacts. As an example, NLP technology can be used for requirement analysis, ambiguity detection, domain concept extraction, and the provision of traceability support across different artifacts.

2.1.3. Deep Learning & Transformers

Deep learning (DL) methods in particular, neural networks which employ sequential or structured representations of code offer a way for AI to go beyond the human developer and into the software development realm. By representing code mathematically (i.e., an "embedding") for the code token, syntax tree, and control/data flow graph, DL can recognize higher-level concepts in big software systems from the patterns it discovers in the software structure. These abstractions allow for the use of these representations for a variety of tasks ranging from suggestions for code completion to code summarization and searching for and synthesizing small portions of code, with the prospect of reducing development labor to improve the reusability and understanding of software.

2.2. Trends in AI-Enabled Software Engineering

The introduction of AI/ML as a part of Software Quality has been basically standard practice now and goes hand in hand with the progressively more advanced tools that are available. Defect prediction, risk assessment and maintenance planning using machine learning have started supplementing and outpacing more traditional approaches to static analysis and heuristic. This indicates a trend that AI is more than just code generating, suggesting AI is a key part of the Software Development Lifecycle where AI can be used to improve software reliability and maintainability at all stages of development.

The use of statistical and deep learning model (NLP) technology is trending increasingly towards creating natural language artefacts (documentation, requirements, comments) from coding activities [8]. Automation at its core is influenced heavily by a mix of these features for example, the activities of document creation (with a possibility of user intervention), code summarization, requirement mapping from the code, and fixing of semantic/syntactic anomalies can all be achieved automatically.

3. AI-Driven Productivity i3n Coding

The Productivity of Coding Using AI Technology; by applying advanced intelligent methods to drive the productivity of Coders via Automating Coding, Identifying Errors, Refactoring, and Understanding the Code, the manual effort to develop Software becomes much less, the resulting productivity of Software Developers is increased, and the Software Development Process is Magnified

3.1. Automated Code Generation Tools

AI-driven automatic code generation tools primarily function as developers assistants by taking up the requirements, templates, or context clues. In this way, developers can escape from the trap of repetitive coding. Additionally, the speed and consistency of the development process are improved by these technologies.

3.1.1. NLP-Based Code Assistants (e.g., GPT-Style Models)

Neural network-based sequence-to-sequence models allow developers to create higher quality code by learning how developers use APIs by training on large amounts of source code[9]. These systems convert natural language description of a task into the appropriate sequence of API calls and reduce the amount of time developers spend writing code for repetitive tasks. The derived API call sequence is based on the semantic intent contained in the user's query. This type of solution reduces the amount of repetitive code that must be written as well as allowing for easier API discovery by providing developing systems with a more complete understanding of how to organize their API calls correctly.

3.1.2. Template and Snippet Generators

The automated snippet generator allows for the quick creation of new software via the retrieval and modification of pre-existing software code from large repositories through the use of analysis of the structural similarities, API patterns, and developer intent to present developers with example templates based on the coding of real-world projects. The automated snippet generator through automation of the processes of fetching and changing code drastically reduces the time of creating boilerplate thus, developers can free up their time to focus on writing the core functionality of their applications. Besides that, automated snippet generation by taking advantage of the well-tested implementation patterns of the previous codebases, guarantees that developers apply the same coding structure when working on different projects.

3.2. AI-Enhanced Code Review

Contemporary methods of code review can be enhanced with machine learning-based detectors that evaluate code changes, thus, the system doesn't have to depend only on manually created rules [10]. A deep model trained on past code changes along with the associated human comments can in effect train itself to figure out which changes are most probably the ones that need an extra check, thereby, the system suggesting reviewers or, in fact,, by giving the user the view of the most questionable code changes.

3.2.1. Bug Pattern Detection

Based on learning, bug detectors search through the source code for repeated bug patterns by recognizing bugs as semantic inconsistencies that have not been resolved, instead of just simple syntactic violations [11]. For this purpose, the systems use neural representations of code tokens and contextual relationships to find the most probable operations, to check if a variable has been wrongly used or to decide if the control flow is abnormal.

Automated model mining frameworks look through past bug fixes to figure out recurring fixing patterns which can be associated with common defect types [12]. Such systems dig into the changed parts of the code both structurally and semantically to find changes that most of the time help to fix a certain class of bugs, for example, null handling, API misuse, or loop boundary issues.

3.2.2. Style, Security, and Standard Compliance

Automated systems for learning-based code evaluation can be of great help in checking if code is in line with the predefined style guidelines and if it follows best practices for maintainability. These automated systems have built a model of standard coding practices for projects after learning about coding conventions from examples in sizable code repositories. This model can then be used to identify any inconsistencies in how things are named, formatted, structured, or used with APIs.

Static analysis tools are an essential aspect of the software development process for enforcing coding style, coding standards, security compliance, and so forth through source code evaluation (without actually running the code) to find and

report any deviations from defined expectations (i.e., "rules"). The static analysis tool can identify programming defects, and potential security risks (e.g., using unsafe coding constructs, or having problems with the logic in the code), as well as provide early identification of code that does not conform to coding convention or coding best practices, therefore, supporting the developer by reducing the likelihood of having to fix coding defects late in the process. A static analyzer highlights not only style inconsistencies and name inconsistencies but also highlights structural problems that should be fixed early on with respect to security relevant violations [13].

3.3. Intelligent Refactoring & Optimization

Using artificial intelligence to intelligently refactor and optimize code to better structure/refactor it, minimize complexity, improve maintainability, increase technical debt/managing the ability to eliminate as much as possible it, thus increasing software quality over the years.

3.3.1. Code Smell Detection

Structural features in programming that are seen as obstructing or old-fashioned in their country of origin are referred to as "code smells." These features have been shown to play a significant role in factors that impact software aerial persistency [14]. Through systematic code analysis, code smell detection can be used to locate elements within the code could have an adverse effect on the code's quality without affecting its functionality. Once code smells have been identified within the code by systematic means, automated methods are available for the improvement of those code smells through the use of development tools or the expertise of developers.

3.3.2. Automated Complexity Reduction

The intelligent refactoring method is an automated way of simplifying code without changing the way it behaves. Whenever the bot finds chances to simplify the code, it makes an inventory of recommended changes that remove repetitions and decrease the total complexity of the code [15]. The proposed changes are delivered as a pull request, which developers can look through and give their consent. Therefore, this automated process allows for continuous improvement in the design quality of software because structural complexity is consistently reduced.

4. AI in Automated Software Testing

AI takes advantage of machine learning, search-based techniques, and intelligent analytics to create, optimize, and detect defects from various sources in order to reduce manual input and increase software testing's effectiveness, scope, dependability, and quality over the course of its whole lifespan.

4.1. Smart Test Case Generation

AI Methods for generating smart test cases generate effective test cases automatically using artificial intelligence (AI) algorithms, so that any application is able to generate a large number of unique use cases while providing maximum functional and fault coverage in less time compared to traditional test case creation methods.

4.1.1. ML-Based Input Generation

Evolutionary and search-based methods are used to develop the different techniques by which test inputs can be automatically generated resulting in an improvement of structural coverage. Employing such ML-inspired and search-based techniques has contributed to lessening the manual work that is involved in the creation of test cases and has also enhanced the fault-revealing potential of test suites.

4.1.2. Model-Based Testing Techniques

Model-based testing (MBT) uses formally specified models or partially defined models regarding the intended behavior of a system under test to help automated test creation and execution. Once created, MBT generates test cases from the models in a consistent way through tool support to be automatically generated and to ensure that the Functional Scenario is fully tested, allowing for a reduction of Manual Intervention during test execution [16]. The MBT Process includes creating a Model or an Abstract Model of the system, determining the criteria for automated Test Generation, generating Abstract Test Sequences and executing them against the System Under Test in a systematic way that aligns with the design of the System.

4.2. Test Suite Prioritization & Optimization

AI-based analysis of the Testing Suite Prioritization and Optimization helps to reorder or filter out test case values as needed or to reduce redundant efforts by analyzing which test cases are likely to identify a fault in the software compared to its execution cost.

4.2.1. Redundancy Reduction

Through redundant test suite reduction, the efficiency of testing may be upgraded such that the number of tests can be decreased while the same level of fault detection is preserved. One method that has been proposed uses genetic algorithms and neural networks together to analyze and optimize test set data [17]. By combining the two forms of optimization neural networks

improve the total number of tests being investigated and the overall testing effort involved in executing those tests as it leads to faster execution and lowers maintenance costs, while at the same time maintaining adequate levels of effectiveness for the tests.

4.2.2. Test Coverage Enhancement

One of the main strategies used by testers to increase coverage in automated software testing is the supplementation of the traditional measures of coverage with ML insights. The additional points of data help to figure out the most efficient way of placing test cases for running. It may be possible to conduct more tests early in the software development cycle that have a bigger impact on an application's total coverage by identifying which test cases, based on their historical pattern and structure, are most likely to reveal faults. Therefore, combining historical defect information with structural information provides an overall view of the test cases that should be executed first, prior to moving to tests that provide a higher degree of risk in the regression test process without sacrificing structural code coverage.

4.3. AI-Driven Bug Detection & Prediction

The detection of software code flaws using a combination of analytics and machine learning algorithms and predicting failure with AI-based software bug detection and prediction technologies helps to improve software development quality by enabling developers to intervene at the earliest possible stage, reducing the amount of time spent maintaining software throughout its lifecycle.

4.3.1. Static Analysis Automation

By analyzing how people have fixed bugs in the past, automation techniques that utilize static analysis able to identify bugs in the future and even potentially recommend a solution for them based on those fixed bugs [18]. The resulting structure allows for representative code changes that a developer could use to address a static analysis warning by providing a recommendation for similar to way a human would correct code. Therefore, using machine learning in this manner allows developers to not only identify static analysis issues but to also be guided towards resolving them as if they were receiving guidance from an experienced developer.

4.3.2. Dynamic Testing with Reinforcement Learning

Dynamic Software Testing methods have the potential to be significantly improved through the employment of Reinforcement Learning (RL) abilities. With the Reinforcement Learning (RL) concept, the testing agent learns from the results of their actions to improve (optimize) the execution of its testing activities [19]. This is the basis for the way that Retecs applies Reinforcement Learning (RL) to constantly adjust and prioritize test case execution for the ever-changing Continuous Integration (CI) framework by factoring in historical execution information, failure rates, and how long it took to run the test cases. The RL agent earns rewards for identifying failures prior to progressing with subsequent tests, and it generates a testing policy based on the testing process that would comprise the greatest quantity of test cases with the highest failure rate.

4.4. Continuous Testing in CI/CD Pipelines

Continuous Testing is a major component of quality assurance to ensure that changes made to source code validated by automated testing before being moved to the next stage in a CI/CD pipeline. Continuous testing allows teams to quickly identify flaws at the earliest phases of the development lifecycle by using automated unit, integration, system, and regression tests, also they indirectly contribute to reducing defect feedback times and to keeping the stability state during fast code deployments. Moreover, the usage of Continuous Testing in the CI/CD pipeline is not a standalone activity. It needs support in the form of monitoring and giving feedback via structured monitoring and feedback loops, the planned and balanced engagement of automation tools, and following best practices like employing test automation frameworks and keeping environment uniformity as prerequisites for software quality assurance being at a high level, releasing frequently, and efficient DevOps workflows.

5. AI-Assisted Deployment and Devops Practices

The automation of the deployment in DevOps through AI-assisted techniques provides the capability to use predictive analytics and intelligent automation as tools for increasing efficiency by providing real-time management capabilities for all infrastructure resources as well as the ability to accelerate software releases with fewer errors and greater reliability.

5.1. AI-Enabled CI/CD Automation

Using AI-powered continuous integration and continuous delivery (CI/CD) Automation enables developers to automate their software development methodology through automatic builds (CI), tests, deployments and monitoring of their work with the use of machine learning and intelligent automation tools that enable improved productivity, decreased error rates and faster software release cycles.

5.1.1. Workflow Automation

Automation of workflows has improved the software development process through greater efficiency. Automated processes allow for the orchestration of all CI/CD pipeline functions (i.e., code integration, building, testing, and deployment) with less manual intervention by developers than previously possible. Hence, automation reduces the time required for software delivery

cycles and eliminates many of the potential sources of code integration errors. In addition, automation is critical to the success of modern DevOps Workplace Philosophy, which necessitates the complete and smooth orchestration of all activities associated with software delivery.

5.1.2. Predictive Build Optimization

Utilizing Machine Learning, Predictive Build Optimization acquires software Change Characteristics and forecasts how changes affect Build Performance in order to enable proactive optimization of Software Build Processes [20]. This information gives Developers and DevOps teams an opportunity to resolve Build Bottlenecks early on, prioritize code changes which have the smallest negative impact on Build Time, and improve their Continuous Integration Processes by minimizing unexpected Slowdowns in their Builds and providing a quicker feedback cycle for their developers [21]. This advanced insight into Build Behavior also offers the opportunity for Intelligence to be incorporated into Job Automation and Resource Allocation in CI/CD Pipeline.

5.2. Intelligent Monitoring & Incident Management

AI-based intelligent monitoring and incident management provide constant monitoring, anomaly detection, automating response, and predictive failure of the various types of systems utilized, improving reliability, decreasing the amount of time that a system is down, and allowing for proactive solutions to software incidents.

5.2.1. Anomaly Detection

Environment monitoring is an essential component of intelligent management systems. Present-day sophisticated and efficient systems rely on anomaly detection to recognize deviations in system performance or log data that can lead to failures or security attacks. To train models for normal behavior and then identify strange actions, the following techniques can be used: unsupervised clustering, statistical analysis, and other machine learning algorithms.

5.2.2. Automated Root Cause Analysis

Automated root cause analysis (RCA) makes use of AI to figure out the source of anomalies or failures of a software system. By using dependency graph analysis, log mining, and probabilistic modelling, RCA technologies can detect the situations that a software system is becoming faulty and then determine which components are the ones causing the system to be faulty. These techniques compare system metrics, event sequences, and historical failure patterns to give the most likely solutions if the problem is fixed based on the data collected by RCA tools; in this way, RCA tools can lower MTTR and raise operational resiliency in CI/CD pipelines.

5.3. Self-Healing Systems & Autonomous Deployment

The following technologies that enable self-healing systems and an automated deployment process make use of Artificial Intelligence to enable automatic detection of issues, initiate a corrective response and maintain system integrity/stability for minimum disruption and maximum efficiency in minimizing human involvement in the operation of software.

5.3.1. Auto-Rollback

The auto rollback ability in deployment workflows gives the ability to automatically roll back deployments to a previously stable version if the deployment encounters any anomalies, degradation in performance as a result of failure or errors detected by health checks. Machine Learning (ML) can provide real-time detection of failure patterns by monitoring applications and reviewing application logs, telemetry data and End-user Quality metrics.

5.3.2. Auto-Scaling Strategies

Predictive auto-scaling refers to the utilization of time series forecasting using machine learning to forecast future workloads in order to provide the best possible resources for the elastic cloud environment [22]. Predictive auto-scaling leverages machine-learning supported algorithms such as support vector machine (SVM) regression to create an estimate of the overall distributed service processing load to determine the amount of back-end servers necessary to fulfill projected demand while minimizing the over-provisioning of resources and improving the overall response time [23].

6. Literature Review

The material analysed in this section is concerned with AI-enabled software engineering-related studies that have been published to date and contains Intelligent Coding Assistance, Automated Testing Methods, Test Optimization, and AI-Assisted Deployment methods.

Huang et al. (2019) presents an overall view of the life cycle of the vulnerability of Smart Contracts from a design, development, testing and support standpoint. This paper's merit is that it offers a category framework for grouping previously created techniques and highlighting areas in need of more investigation. However, there is limited critical evaluation of the efficacy of these methods when applied to various Blockchain Platforms. Thus, a rigorous comparative analysis using quantitative security metrics would help make this work more relevant to Blockchain Engineers [24].

Tao, Gao and Wang (2019) focuses on an innovative area of quiet research, which is the quality of AI software validation, and presents a practical metamorphic testing method for image recognition systems as a solution. However, the study only considers a few functional features, so the behaviours of AI systems such as bias or explain ability that have not been considered are still largely unexplored. There is a potential for the next research to broaden the scope of testing frameworks for different AI models and various deployment scenarios [25].

Morán et al. (2018) presents MRTest as a revolutionary method to test Big Data programs that have dynamically changing infrastructures, a situation where faults that have not been detected for a long time in stable environments are most frequently referred to. The paper effectively shows the manifestation of the testing process being automated and adaptable. Nevertheless, the assessment is largely limited to design faults, thus the paper does not raise performance and scalability issues of big data in production environments. It would be a lot more convincing and useful if the paper included some actual implementation scenarios to corroborate the approach [26].

Panichella, Kifetew and Tonella (2018) introduces DynaMOSA, which is a new automated testing generation many-objective optimization approach that has demonstrated through experimentation to provide superior results when compared against previous state-of-the-art methods for the areas of code coverage and mutation testing. The focus of the current research is limited to Java classes and to specific coverage solutions, therefore there are limitations for generalization across other programming languages and testing environments [27].

Shahin, Ali Babar and Zhu (2017) Identified tools, practices, and barriers to industrial use through this systematic review provide a good overview of what has been accomplished with continuous deployment, continuous integration, and continuous delivery, as well as pinpoint areas where knowledge about the influence of the organization's environment (e.g., size, industry) is lacking. Quantitative comparisons of the success/failure of these practices between companies would help give practitioners better direction on implementing these types of systems [28].

Gay, Rayadurgam and Heimdahl (2017) presents an entirely new approach to the creation of test oracles that would effectively eliminate false positives in testing non-deterministic systems. This new approach provides a way to align the model used to create the oracle with the behaviour of the SUT while keeping the amount of variability reasonable [29].

Table I summarizes the main studies of AI-driven software development that includes the research focus, approaches used, major findings, limitations, and future research directions in coding, testing, and deployment domains.

Table 1. Comparative Analysis of AI-Driven Software Engineering Approaches in Coding, Testing, and Deployment

Reference	Study On	Approach	Key Findings	Challenges / Limitations	Future Directions
Huang et al. (2019)	Smart Contract Security	Lifecycle-based literature review (design, implementation, testing, monitoring)	Provides a structured classification of smart contract vulnerabilities and mitigation techniques	Limited critical comparison of methods across different blockchain platforms	Comparative analysis using quantitative security metrics across multiple blockchain ecosystems
Tao, Gao & Wang (2019)	AI Software Testing and Quality Validation	Metamorphic testing applied to image recognition systems	Demonstrates feasibility of systematic testing for AI functional features	Narrow focus on limited AI behaviors; neglects bias, fairness, and explainability	Expansion of testing frameworks to diverse AI models, behaviors, and deployment environments
Morán et al. (2018)	Big Data Software Testing	Automated infrastructure-simulation–based testing using MRTest	Effectively detects design faults hidden in stable testing environments	Evaluation limited to design faults; performance and scalability not addressed	Validation in real-world production environments and inclusion of performance-related testing
Panichella et al. (2018)	Automated Test Case Generation	Many-objective optimization using DynaMOSA	Achieves higher code coverage and mutation detection than prior methods	Restricted to Java programs and specific coverage criteria	Adaptation to other programming languages and broader testing scenarios

Shahin et al. (2017)	Constant Deployment, Delivery, and Integration	Systematic literature review with thematic analysis	Identifies tools, practices, and barriers affecting continuous practices adoption	Limited insight into organizational context and lack of quantitative comparisons	Empirical, cross-organizational studies assessing effectiveness of continuous practices
Gay et al. (2017)	Test Oracle Generation for Non-Deterministic Systems	Model steering framework to reduce false positives	Significantly improves oracle–SUT conformance while preserving fault detection	Evaluation scope limited to specific system types	Large-scale validation and integration into industrial testing pipelines

7. Conclusion and Future Work

Artificial Intelligence (AI) influences Smart Software Development in different ways, such as enhancing coding productivity, automating software testing, deployment, etc. In particular, using deep learning, machine learning, and natural language processing (NLP), AI technologies are enabling the automatic generation of source code, Intelligent Code Review, Code Refactoring, Defect Prediction, Optimized Test Execution. Continuous Integration and Continuous Deployment (CI/CD) pipelines eventually incorporate Artificial Intelligence (AI) to enable uninterrupted build, monitor, and incident management operations. The use of AI for self-healing and autonomous deployment strategies has enabled significant reductions in manual effort required for software delivery, as well as reduced time spent developing software; therefore enhancing the quality and reliability of software. Still, issues remain that require attention, like the scaling and understanding of AI Models, as well as the elevating of organizations to the level where they can seamlessly integrate incorporating AI into their software development procedures.

Future work focused on the usage of models based on explainable AI in order to have more trust in the decisions of automated software engineering. Moreover, these should be substantiated by several big industrial validations across diverse fields so that their extendibility and strength can be assessed. It is still a challenge to incorporate morally guided AI components, bias recognition, and security-aware automated processes into SDLC tools. Moreover, the next step in the evolution of software development ecosystems is probably the emergence of adaptive learning systems that will be able to change with codebases and deployment environments, thus becoming autonomous, resilient, and intelligent.

References

[1] K. Shankari and R. Thirumalaiselvi, “A Survey on Using Artificial Intelligence Techniques in the Software Development Process,” vol. 4, no. 12, pp. 24–33, 2014.

[2] V. Sresth, S. P. Nagavalli, and A. Srivastava, “Artificial Intelligence in Software Engineering : Transforming Development Paradigms in Financial Services through Machine Learning Innovations,” vol. 02, no. 01, pp. 50–62, 2019.

[3] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, “A Survey of Machine Learning for Big Code and Naturalness,” *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–37, Jul. 2017, doi: 10.1145/3212695.

[4] M. A. Umar and C. Zhanfang, “A Study of Automated Software Testing : Automation Tools and Frameworks,” *Int. J. Comput. Sci. Eng.*, 2019, doi: 10.5281/zenodo.3924795.

[5] R. H. Kulkarni and P. Padmanabham, “Integration of artificial intelligence activities in software development processes and measuring effectiveness of integration,” *IET Softw.*, vol. 11, no. 1, pp. 18–26, Feb. 2017, doi: 10.1049/iet-sen.2016.0095.

[6] H. V. Gamido and M. V. Gamido, “Comparative Review of the Features of Automated Software Testing Tools,” *Int. J. Electr. Comput. Eng.*, vol. 9, no. 5, p. 4473, Oct. 2019, doi: 10.11591/ijece.v9i5.pp4473-4478.

[7] V. M. L. G. Nerella, “Automated cross-platform database migration and high availability implementation,” *Turkish J. Comput. Math. Educ.*, vol. 9, no. 2, pp. 823–835, 2018.

[8] A. Leclair, S. Jiang, and C. Mcmillan, “A Neural Model for Generating Natural Language Summaries of Program Subroutines,” 2019.

[9] X. Gu, H. Zhang, D. Zhang, and S. Kim, “Deep API learning,” in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA: ACM, Nov. 2016, pp. 631–642. doi: 10.1145/2950290.2950334.

[10] J. K. Siow, C. Gao, L. Fan, S. Chen, and Y. Liu, “CORE : Automating Review Recommendation for Code Changes,” *arXiv*, 2019.

[11] A. Habib and M. Pradel, “Neural Bug Finding : A Study of Opportunities and Challenges,” *Articial Intell. Robot.*, 2019.

[12] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, “An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation,” *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 4, pp. 1–29, Oct. 2019, doi: 10.1145/3340544.

[13] E. E. Sajan, Y. Zhang, and L. Cheng, “Static Analyzers and Potential Future Research Directions for Scala : An Overview,” *arXiv*, 2019, doi: 10.48550/arXiv.1905.04752.

[14] S. Singh and S. Kaur, “A systematic literature review : Refactoring for disclosing code smells in object oriented software,”

Ain Shams Eng. J., vol. 9, no. 4, pp. 2129–2151, 2018, doi: 10.1016/j.asej.2017.03.002.
[15] V. Alizadeh, M. A. Ouali, M. Kessentini, and M. Chater, "RefBot: Intelligent Software Refactoring Bot," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, Nov. 2019, pp. 823–834. doi: 10.1109/ASE.2019.00081.
[16] P. Daca, T. A. Henzinger, W. Krenn, and D. Nickovic, "Compositional specifications for ioco testing," in *Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation, ICST 2014*, 2014. doi: 10.1109/ICST.2014.50.
[17] I. Hooda and R. S. Chhillar, "Test Case Optimization and Redundancy Reduction Using GA and Neural Networks," vol. 8, no. 6, pp. 5449–5456, 2018, doi: 10.11591/ijece.v8i6.pp5449-5456.
[18] J. Bader, A. Scott, M. Pradel, and S. Chandra, "Getafix : Learning to Fix Bugs Automatically," 2016.
[19] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration," 2018, doi: 10.1145/3092703.3092709.
[20] M. Tufano, H. Sajnani, and K. Herzig, "Towards Predicting the Impact of Software Changes on Building Activities," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, IEEE, May 2019, pp. 49–52. doi: 10.1109/ICSE-NIER.2019.00021.
[21] S. Garg, "Predictive Analytics and Auto Remediation using Artificial Inteligence and Machine learning in Cloud Computing Operations," *Int. J. Innov. Res. Eng. Multidiscip. Phys. Sci.*, vol. 7, no. 2, 2019, doi: 10.5281/zenodo.15362327.
[22] A. Kushwaha, P. Pathak, and S. Gupta, "Review of Optimize Load Balancing Algorithms in Cloud.," *Int. J. Distrib. Cloud Comput.*, vol. 4, no. 2, p. 1, 2016.
[23] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Efficient resource provisioning for elastic Cloud services based on machine learning techniques," *J. Cloud Comput.*, vol. 8, no. 1, p. 5, Dec. 2019, doi: 10.1186/s13677-019-0128-9.
[24] Y. Huang, Y. Bian, R. Li, J. L. Zhao, and P. Shi, "Smart Contract Security: A Software Lifecycle Perspective," *IEEE Access*, vol. 7, pp. 150184–150202, 2019, doi: 10.1109/ACCESS.2019.2946988.
[25] C. Tao, J. Gao, and T. Wang, "Testing and Quality Validation for AI Software–Perspectives, Issues, and Practices," *IEEE Access*, vol. 7, pp. 120164–120175, 2019, doi: 10.1109/ACCESS.2019.2937107.
[26] J. Morán, A. Bertolino, C. de la Riva, and J. Tuya, "Automatic Testing of Design Faults in MapReduce Applications," *IEEE Trans. Reliab.*, vol. 67, no. 3, pp. 717–732, 2018, doi: 10.1109/TR.2018.2802047.
[27] A. Panichella, F. M. Kifetew, and P. Tonella, "Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets," *IEEE Trans. Softw. Eng.*, vol. 44, no. 2, pp. 122–158, 2018, doi: 10.1109/TSE.2017.2663435.
[28] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.
[29] G. Gay, S. Rayadurgam, and M. P. E. Heimdahl, "Automated Steering of Model-Based Test Oracles to Admit Real Program Behaviors," *IEEE Trans. Softw. Eng.*, vol. 43, no. 6, pp. 531–555, 2017, doi: 10.1109/TSE.2016.2615311.
[30] Mamidala, J. V., Enokkaren, S. J., Attipalli, A., Bitkuri, V., Kendyala, R., & Kurma, J. (2023). Machine Learning Models Powered by Big Data for Health Insurance Expense Forecasting. *International Research Journal of Economics and Management Studies IRJEMS*, *2*(1).
[31] Nadella, V. M. (2023). Zero Trust Architecture for Telecom Operations. *International Journal of Emerging Research in Engineering and Technology*, *4*(3), 115-129.
[32] Bitkuri, V., Kendyala, R., Kurma, J., Enokkaren, S. J., & Mamidala, J. V. (2023). Forecasting Stock Price Movements With Deep Learning Models for time Series Data Analysis. *Journal of Artificial Intelligence & Cloud Computing. SRC/JAICC-531. DOI: doi. org/10.47363/JAICC/2023 (2)*, *489*, 2-9.
[33] Nadella, V. M. (2023). Anomaly Detection and Fault Prediction using ML in Telecom Operations. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(3), 134-143.
[34] Kosaraju, P., & Nadella, V. M. (2022). Security and Privacy in IoT Ecosystems. *Universal Library of Engineering Technology*, (Issue).
[35] Singh, A. A. S. S., Mania, V., Kothamaram, R. R., Rajendran, D., Namburi, V. D. N., & Tamilmani, V. (2023). Exploration of Java-Based Big Data Frameworks: Architecture, Challenges, and Opportunities. Journal of Artificial Intelligence & Cloud Computing, 2(4), 1-8.
[36] Routhu, K. K. (2023). AI-driven succession planning in Oracle HCM Cloud: Building resilient leadership pipelines through predictive analytics. *International Journal of Science, Engineering and Technology*, *11*(5).
[37] Tamilmani, V., Namburi, V. D., Singh Singh, A. A., Maniar, V., Kothamaram, R. R., & Rajendran, D. (2023). Real-Time Identification of Phishing Websites Using Advanced Machine Learning Methods. *Available at SSRN 5837142*.
[38] Routhu, K. K. (2023). AI-driven succession planning in Oracle HCM Cloud: Building resilient leadership pipelines through predictive analytics. International Journal of Science, Engineering and Technology, 11(5). https://doi.org/10.5281/zenodo.17292018

[39] From Fragmentation to Focus: The Benefits of Centralizing Procurement. (2023). International Journal of Research and Applied Innovations, 6(6), 9820-9833. https://doi.org/10.15662/

[40] Routhu, K. K. (2023). Embedding fairness into the digital enterprise, data driven DEI strategies with Oracle HCM Analytics. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *9*(8), 266-274.

[41] Routhu, K. K. (2023). AI-driven skills forecasting in Oracle HCM Cloud: From static competencies to predictive workforce design. *International Journal of Science, Engineering and Technology*, *11*(1).

[42] Padur, S. K. R. (2023). AI-Augmented Enterprise ERP Modernization: Zero-Downtime Strategies for Oracle E-Business Suite R12. 2 and Beyond. *Available at SSRN 5605510*.

[43] Routhu, K. K. (2022). From Case Management to Conversational HR: Redefining Help Desks with Oracle's AI and NLP Framework. *International Journal of Science, Engineering and Technology*, *10*(6).

[44] Vattikonda, N., Gupta, A. K., Polu, A. R., Narra, B., Buddula, D. V. K. R., & Patchipulusu, H. H. S. (2022). Blockchain Technology in Supply Chain and Logistics: A Comprehensive Review of Applications, Challenges, and Innovations. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(3), 72-80.

[45] Attipalli, A., BITKURI, V., Mamidala, J. V., Kendyala, R., & KURMA, J. (2022). Empowering Cloud Security with Artificial Intelligence: Detecting Threats Using Advanced Machine learning Technologies. *Available at SSRN 5741263*.

[46] Padur, S. K. R. (2022). Intelligent resource management: AI methods for predictive workload forecasting in cloud data centers. *J. Artif. Intell. Mach. Learn. & Data Sci*, *1*(1), 2936-2941.

[47] Nadella, V. M. (2022). Digital Twins for Predictive Network Management and System Simulation. *International Journal of AI, BigData, Computational and Management Studies*, *3*(3), 100-111.

[48] Routhu, K. K. (2022). From RFID to Geofencing: IoT-Enabled Smart Time Tracking in Oracle HCM Cloud. *International Journal of Science, Engineering and Technology*, *10*(4).

[49] Nadella, V. (2019). Extracting road traffic data through video analysis using automatic camera calibration and deep neural networks.

[50] Polam, R. M., Kamarthapu, B., Kakani, A. B., Nandiraju, S. K. K., Chundru, S. K., & Vangala, S. R. (2022). Data Security in Cloud Computing: Encryption, Zero Trust, and Homomorphic Encryption. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(4), 31-41.

[51] Padur, S. K. R. (2022). AI augmented platform engineering, transforming developer experience through intelligent automation and self optimizing internal platforms. *International Journal of Science, Engineering and Technology*, *10*(5), 10-5281.

[52] Kosaraju, P. , & Nadella, V. M. (2021). Quality of Experience (QoE) and Network Performance Modelling for Multimedia Traffic. Journal of Artificial Intelligence and Big Data, 1(1), 1-13. https://doi.org/10.31586/jaibd.2021.1358.