Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 4, Issue 4, 2026

AI-DEVGUARD: AN ARTIFICIAL INTELLIGENCE FRAMEWORK FOR AUTOMATING AND SECURING THE SOFTWARE DEVELOPMENT LIFE CYCLE

Sherbano Saleem[*1], Dr. Kashif Laeeq[2], Dr. Muhammad Asad Abbasi[3]

[*1]*Department of Computing, (FCIT) Indus University, Karachi, Pakistan*
[2]*Department of Computer (CCSIS), Institute of Business Management (IoBM), Karachi, Pakistan*
[3]*Benazir Bhutto Shaheed University, Karachi*

[*1]sherbano.saleem@indus.edu.pk, [2]kashif.laeeq@iobm.edu.pk, [3]muhammad.asad@bbsul.edu.pk

DOI: https://doi.org/10.5281/zenodo.19452079

Keywords
Artificial Intelligence, Software Development Life Cycle, AI-driven framework, AI-DevGuard, DevOps pipelines.

Article History
Received: 11 February 2026
Accepted: 21 March 2026
Published: 07 April 2026

Corresponding Author: *

Abstract

The innovation of Artificial Intelligence (AI) has continued to positively impact numerous fields, including software development. In this research, we present the AI-DevGuard, an AI powered architecture designed to automate and protect subdivisions of the software development life cycle (SDLC) using intelligent tools and techniques. Using AI-DevGuard, software development teams will be able to automate the detection of bugs, improve the precision of the software, construct the software using an automated process, and improve the protection of the software through the early detection of bugs. This research will perform an in-depth analysis of the potential of AI-DevGuard in reducing the effect of human negligence, facilitating the shortening of time frames, and increasing the security in the development process. Here, we fully describe the unique plug and play design of AI-DevGuard and its compatibility to leading development tools that allows it to seamlessly integrate into the existing development tools of today's DevOps.

I. INTRODUCTION

AI is already part of the SDLC (Software Development Life Cycle) and changing the software maintenance and development industry. SDLC AI applications are not theoretical and stand-alone but practical and integrated cross-phase applications for the first time. AI is automating coding functions, streamlining systems design and deployment, and enhancing AI-assisted decision-making throughout the software development process [1].

AI systems built on deep learning, predictive analysis, NLP and machine learning techniques offer real-time solutions to some complex software engineering problems. Software engineering confront large attack vectors due to challenges of scalability, timelines, errors, quality assurance, and security concerning software development. Consider problems solved by AI systems like OpenAI codex and GitHub Copilot, which lessen manual coding, and AI-driven testing systems, which improves the speed and reliability of continuous integration pipelines. Moreover, the software development industry confronts integration issues related to the AI paradox of bias, accountability, explainability, job loss, and, most importantly, profit. These are the main issues that stands from AI integration through SDLC and it is very essential to balance such issues on time.

This research examines the impact of the

integration of AI on the workflows of different software development methodologies and the effects of AI on software development. In particular, the study focus on the ethical, technological, and collaboration challenges of AI. It will investigate the practical aspects of Artificial Intelligence in software engineering, the practical perspective of developers on Artificial Intelligence, and the discourse surrounding the use and, more importantly, the abuse of AI in engineering software. This will determine AI's relevance and practical limits in software engineering and help identify potential gaps. AI is relevant to software engineering in all its infinities. Unlike the agile methodologies of the past twenty years, AI will enable final intelligent automation to complete software development, providing intelligent automation to handle seamless shifts and transitions in the development process [2].

One of the fundamental reasons for the rapid transformative changes in the discipline of development is an integration of AI in the software development life cycle (SDLC). In figure 1, the classical SDLC is presented as a sequence of procedures. Each step in the sequence is a software development task and process carried out manually by a human. There is no automation.

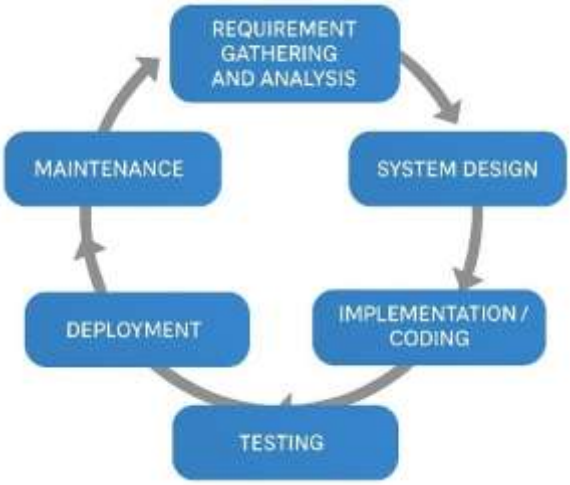

Figure 1: Traditional Software Development Lifecycle

In the past, these methods may have worked, but they now fail to provide quick, scalable, and seamless software solutions. AI came to assist in the automation, velocity, and the reliability enhancement across the various phases in the software development lifecycle.

With the advances in AI and specifically ML, DL, and NLP, and analytics, the automation of time-consuming and monotonous tasks is now a reality [3]. Consider GitHub Copilot and OpenAI Codex, for example, which produce syntactically accurate code in response to basic natural language directives. Similarly, AI-enabled software testing technologies predict gaps in test coverage, produce test cases, and detect anomalies to enhance reliability and quality [4].

Undoubtedly, there are benefits of employing AI, but there are also unanswered questions and problems that it brings about. These include a lack of transparency, justice, accountability, the disproportionate impacts, biases and inequalities of AI decision-making, reliance due to automation, and organizational transformation agility. In the lack of a clear position, the explainability of AI, the ethics of AI-induced decision-making, the rights of the data subjects, and the responsibilities of the AI developer are focal points of the discussion. The need for radical AI tools standardization, ethical calibration, and oversight is more imperative in high-stakes industries like healthcare and finance. In the discipline of project management, forecasting

systems for predicting project delays, in the allocation of resources, and during the real-time monitoring and risk management phases, there are predictors in AI systems. The ethical risks, need for interpretability, and developer's role are also overlapping and unresolved issues. The deployment of AI tools in high-stake industries requires the establishment of a solid ethical guideline [3].

II. LITERATURE REVIEW

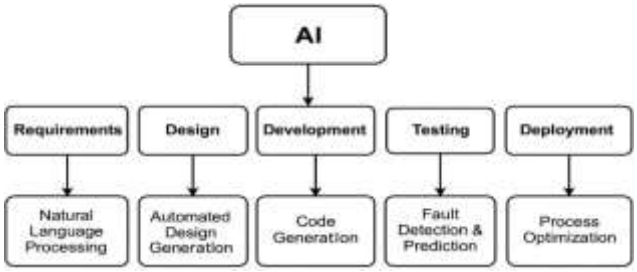

Figure 2: Influence of AI across SDLC phases

The Software Development Lifecycle (SDLC) is a systematic approach to the planning, development, testing, and deployment of an information system. A lot of traditional SDLC strategies and approaches as illustrated in figure 2 are being improved, or in some cases, completely innovated thanks to the scale and complexity of contemporary software systems and the use of Artificial Intelligence. AI implementation across all SDLC phases is improving efficiency, precision, and flexibility. This part describes the various roles of AI in process automation, quality assurance, security and optimization across the entire lifecycle of SDLC [5].

A. Challenges in traditional SDLC

Challenges in traditional SDLC several reasons explain the slow, low quality, and expensive production of software products in traditional SDLC.

1) ***Failure of Timely Completion and Manual Inefficiency:*** Tasks in the SDLC like requirement elicitation, coding, and testing are extensively manual. This means they will be slow and likely to have errors. These are a result of oversight in requirement analysis, coding bugs, or tests that miss edge case scenarios, to cite a few examples. [6].

2) ***Requirements Change Continuously***: During planning, the requirements will change most of the time. The traditional models have a hard time dealing with this. The result of this is software that has features that are misaligned and leads to frustration.

3) ***Weak Security:*** Due to how security measures are implemented at the final stages of the software engineering life cycle, it becomes more expensive and more difficult to close security gaps [7].

4) ***Timeline and resource constraints:*** With the limited budget, time, and people's expertise, the development life cycle will not be fully accomplished, affecting overall quality.

B. Ways to Improve the SDLC Using AI Artificial intelligence

Artificial intelligence technologies help to break traditional constraints at some points in the SDLC by Intelligent Process Automation and Advanced Analytics.

1) ***AI in Requirement Engineering:*** Using NLP tools to extract requirements in structured formats from unstructured needs documents, emails, and meeting notes. AI models automatically identify ambiguities or contradictions in user stories, facilitating the interaction of clients and developers.

2) ***AI in Software Design*:** Machine learning models suggest design patterns, while AI systems analyze and critique designs of software architectures before they are built, thus minimizing potential errors. [8].

3) ***AI in Code Generation:*** Generative AI such as OpenAI Codex and GitHub Copilot turns natural language instructions into computer code. This reduces development time and helps to eliminate repetitive coding tasks, all of which are especially helpful for beginner developers.

4) ***AI in Testing and Debugging:*** AI tools designed for testing can build and run test cases without human assistance. Regression failures or bugs are predicted using AI technique in a training model with historical data. For example, Google uses AI-based testing systems that find critical failures that were missed in later testing, using historical bug data, and log data of a system in production. [9].

5) ***AI in Maintenance and Monitoring:*** AI analyses logs and patterns in usage of a system tools to predict operational failures. He or she recommends or suggests bypass in a workflow, or wise optimization [6].

C. *Security enhancements through AI in SDLC*

AI embedding security from the start of development is crucial.

1) ***Threat Detection and Prevention***: AI integrated tools can find vulnerabilities in the code repositories and can suggest fixing. SonarQube tools, as an example, uses AI to identify security gaps while a developer is writing code.

2) ***Secure Authentication***: AI reduces the risk of unauthorized access in the development and testing phases by incorporating behavioral, user profile, and biometric authentication

3) ***Intelligent Access Control:*** AI technologies assist in role-based access control by analyzing users' roles and access patterns, automatically detecting unusual behaviors and potential insider threats.

III. ARTIFICIAL INTELLIGENCE IN SOFTWARE DEVELOPMENT

The automation of the Software Development Life Cycle (SDLC) can be achieved using computer algorithms such as AI, Deep Learning (DL), Machine Learning (ML), Natural Language Processing (NLP), and other intellectual capabilities [9]. AI can help automate or even fully automate human cognitive functions like code writing, detecting errors, analyzing software requirements, and testing software. AI helps companies work faster and enhances the quality of code, minimizes the time taken to develop software, and helps in the assurance of sound decision-making.

A. AI-Powered Tools and Workflows

AI creates a new era in software development by improving every aspect of the SDLC. Below shows the evolution from traditional to present and future AI tools:

Table 1: Traditional to Present and Future AI tools

SDLC Stage	Old Tools	Modern AI Tools	Next-Gen AI Tools
Requirement Analysis	JIRA, IBM DOORS, Microsoft Visio	ChatGPT, Requiro, OpenAI Codex	NLP-driven interview bots, Automatic SRS generation
Design	UML, Microsoft Visio, Adobe XD	Uizard, Sketch2Code, DeepCode	AI-generated architectures, Real-time consistency validation
Development	Eclipse, Visual Studio, IntelliJ IDEA	GitHub Copilot, CodeWhisperer	Fully automated IDEs
Testing	Selenium, JUnit, TestNG	Applitools, DeepCode, Test.ai	Autonomous testing, Self-healing scripts
Deployment	Jenkins, Docker, Kubernetes	Harness, AIOps	Self-healing pipelines, AI-managed zero-downtime deployment
Maintenance	Bugzilla, ServiceNow, Splunk	IBM Watson AIOps, New Relic	Predictive maintenance, Self-repairing systems

1) Requirement Analysis

- **Old Tools**: JIRA, IBM DOORS, Microsoft Visio
- **Modern AI Tools**: ChatGPT, Requiro, OpenAI Codex
- **Next Gen AI Tools**: NLP-driven stakeholder interview bots, automatic SRS creation

AI employs NLP to parse unstructured emails and meeting transcripts to generate Software Requirement Specifications (SRS). Upcoming AI systems may independently execute stakeholder interviews, resolve ambiguities regarding the requirements, and generate necessary documentation.

2) Design

- **Old Tools:** UML, Microsoft Visio, Adobe XD
- **Modern AI Tools:** Uizard, Sketch2Code, DeepCode
- **Next Gen AI Tools:** AI-generated architectures, real-time consistency validation

Sketch2Code uses AI to transform UI design to functional code. Future advancements will focus on real-time specialized architectural consistency and optimization [5].

3)Development

- **Old Tools**: Eclipse, Visual Studio, IntelliJ IDEA
- **Modern AI Tools**: GitHub Copilot, CodeWhisperer
- **Next Gen AI Tools**: Fully automated ide's

GitHub Copilot analyzes code and suggests fixes for bugs. Future ide's will sense developer intent at a high enough level to self-correct and optimize code dynamically [7].

4) Testing

- **Old Tools**: Selenium, JUnit, TestNG
- **Modern AI Tools**: Applitools, DeepCode, Test.ai

- **Next Gen AI Tools**: Fully autonomous testing, self-healing scripts

AI technology provides automated regression testing and predictive maintenance. Future testing will repair scripts automatically and test violations without human involvement [6].

5) Deployment

- **Old Tools**: Jenkins, Docker, Kubernetes
- **Modern AI Tools**: Harness, AIOps
- **Next Gen Tools**: Self-healing pipelines, AI-managed zero-downtime deployment

AI-integrated CI/CD platforms like Harness predict potential issues and recommend changes using machine learning. Pipelines of the future are expected to self-diagnose and auto-resolve deployment failures [5].

6) Maintenance

- **Old Tools**: Bugzilla, ServiceNow, Splunk
- **Modern AI Tools**: AIOps platforms (e.g., IBM Watson AIOps, New Relic)
- **Next Gen Tools**: Predictive maintenance, self-repairing systems

AI aids post-deployment monitoring and failure prediction. Predictive models analyze logs and performance metrics to recommend automated issue resolution and resolution automation [6].

B. How AI Tools Overcome limitations of Traditional Tools

Traditional software development tools have served the industry for decades. JIRA for requirement management, UML for design, Eclipse for development, and Selenium for testing. However, the fast-paced world of AI-integrated software development presents new challenges. Reimagine the possibilities these tools could offer with AI [4]. Some of the most significant limitations of traditional tools are:

- **Manual Effort**: Most traditional tools require extensive human intervention, from writing code to generating test scripts and deployment configurations.
- **Lack of Intelligence**: Rule based tools and systems do not manage and learn developer behaviors and mistakes. Thus, they will never instill the proactive mentality.
- **Time-Consuming**: Bug detection, requirement validation, and regression testing, especially in large codebases will always be time consuming
- **Limited Scalability**: They are not optimized to the scale, complexity and speed of modern DevOps environments [10].

However, AI-based tools do to these systematically:

- **Automation and Speed**: Tools like GitHub Copilot, OpenAI Codex, Requiro and others can auto-generate code even the test cases and assist requirement engineering through NLP based stakeholder interaction.
- **Predictive Capabilities**: Systems like IBM Watson AIOps and others can assess performance logs and analyze the behavioral patterns to anticipate and even recognize issues before they arise and affect the system. system.
- **Continuous Improvement** AI models develop and learn from historical data and user interactions, thereby become more flexible to evolve and enhance the suggested recommendations over time.
- **Seamless Self-Healing and Smart Operations**: AI tools offer great functions in the SDLC such as test scripts that fix themselves, and auto adapting pipelines.

C. Quantitative Impact of AI Tools on Software Development

Following previous discussions on AI-Powered Tools and Workflows, this section aims to present evidence to which AI tools reduce the time development takes (in person-hours) and the associated costs from case studies of real world software development projects that incorporate current.

Table 2: AI Tools and Their Impact across SDLC Phases

AI Tool	SDLC Phase	Time Saving (%)	Cost Saving (%)
ChatGPT	Requirements Engineering	30%	20%
Requiro	Requirements Engineering	25%	18%
OpenAI Codex	Code Development	40%	35%
Uizard	UI/UX Design	35%	30%
Sketch2Code	UI Design	30%	25%
DeepCode	Code Review & Debugging	25%	20%
GitHub Copilot	Code Development	55%	38%
CodeWhisperer	Code Development	50%	35%
JetBrains AI Assistant	Code Development	45%	32%
Applitools	Testing	50%	40%
Test.ai	Testing	50%	42%
Harness	Deployment	20%	15%
AIOps	Monitoring and Maintenance	35%	28%

Table 2 shows data that indicates how AI tools do much more than just automate monotonous tasks. They also improve efficiency. For example, AI tools like GitHub Copilot and CodeWhisperer more than 50% shorten coding time and Applitools and Test.ai take off 42% in costs by removing the time spent on manual testing with AI-based testing tools. Also, ChatGPT and Requiro help automate documentation generation and user story creation, hastening and simplifying the requirements gathering step [11].

AI design tools Uizard and Sketch2Code allow UI teams to quickly prototype and significantly reduce the cost of designing the front end of the application. These statistics support the statement that the use of AI in software projects leads to a considerable decrease in cost and a decrease in person-hours, as demonstrated. This also improves efficiency and project management predictability

D. I Tech in Software Development as A System

i. Open Desk and Collaborative Artificial Intelligence Workspaces

Open Desk is a collaborative tool designed for public service administrators and provides a digital workspace that includes a word examiner, a project management system, video calls in real-time, and a system for secure communications. AI in Open Desk's collaborative tools enhances inter-groups collaboration by streamlining processes, forecasting task prioritization compared to others and historical data, and locating imbalances in a project's requirements. This work distribution improvement in collaborative AI focuses on software development bottom lines. [12].

ii. Group-Based Development Tools

AI Group Development Tools AI tools in collaboration with GitHub Copilot, CodeGPT suggest coding snippets, perform automatic code reviews, and identify and resolve merge conflicts

which will greatly enhance collaboration among distributed teams. These tools help in analyzing code, which positively impacts the quality of integration and addresses integration issues as well. AI also aids in improving the development workflow by predicting analytic bottlenecks in development, thus enabling project teams to formulate proactive strategies for on-time completion [12].

iii. Business and Open Code Platforms

Integrating AI into business and open-source coding platforms is improving code quality and increasing productivity. One example is Salesforce's AI tool, Agent Force 360, which automates task completion, manages customer relations, and provides business insights. In open-source environments, Desk's AI tools automate code review, identify bugs, and offer suggestions to improve code, thus fostering collaboration and cooperative development. They enhance and tighten software solution development and provide tools to change approaches to software development and provide tools to change approaches to software development [12].

iv. Security Tools: OWASP and AI Security

In this stage of software design, the primary focus needs to be on the best way to secure large language models (LLMs) and Generative AI, all the while implementing and maintaining a security guidance plan as the threat landscape for these AI systems evolves. OWASP AI Security Project encapsulates security controls, threat modeling, and vulnerability assessments during the entire lifecycle of the AI. Without these in application development, teams will be blind to the compliance risks of enterprise AI applications and will be skipped by the OWASP target frameworks [13].

V. Powerful AI-Driven Coding Systems

The following generation of AI-driven coding systems revolutionizes software development. Tools like Devin and Augment Code advance to not only automate coding and re-coding software development tasks but also assist in real-time bug identification and bug fixing. They examine a wide variety of codebases and offer intelligent, contextual recommendations, which accelerates the development process while minimizing errors. Development teams streamlining these AI systems will enhance the quality of their code and the speed at which it is delivered while abstracting to tackle more high-order complex problems.

IV. PROPOSED FRAMEWORK, AI-DEVGUARD

This chapter presents the first research proposal on the first version of AI-DevGuard. The first AI-DevGuard version automates aspects of the software development life cycle (SDLC) intelligently and integrates automation. AI-DevGuard incorporates machine learning, natural language processing, and code-analysis engines to assist development in requirement engineering, code generation, testing, and debugging [14].

AI-DevGuard incorporates the following key functions:

- Retention of secured codes.
- Refactoring and optimizations of codes.
- Validation of requirements through natural language processing.
- Generation and interpretation of AI-driven test cases.

Through AI-DevGuard's trained AI models, developers learning patterned secured codes, core recommendations, and innovative shortcuts found in big open-sourced codes to proprietary codes. AI-DevGuard is designed modulo, and it identifies interop/adaptability, and cloud native [54]. The development cycle sees AI used considerably not merely as enhancing productivity, but also as a supportive and reliable resource in aligning every development phase with required and best practice secured codes AI constructed and imposed controlling limited automation. AI-DevGuard contributes to enhanced software delivery, improved decision making, and reduced human error [15].

A. System Architecture

AI-DevGuard architecture consists of four primary layers:

1. **User Interaction Layer:** Developers interact with the system via intelligent IDEs (e.g., GitHub Copilot, TabNine) that gives suggestions in real time and auto-completions that are on context.
2. **AI Engine Layer:** This section contains the models for code suggestion, bug predicting, and test case generation which are fine-tuned for reinforcement learning and historical data [46].
3. **Security and Compliance Layer:** AI models identify code vulnerabilities, using threat intelligence dataset and respond by either generating alerts or remediation patches [16].
4. **Monitoring and Feedback Loop:** Performed both automated and manual testing to measure the criteria and ensure the functional and non-functional requirements were satisfied. [13].
5. **Testing:** Performed both automated and manual testing to measure the criteria and ensure the functional and non-functional requirements were satisfied.
6. **Requirement Validation:** This is the last check to ensure that all post-development requirements are met. This helps close the loop between requirement engineering and delivery.

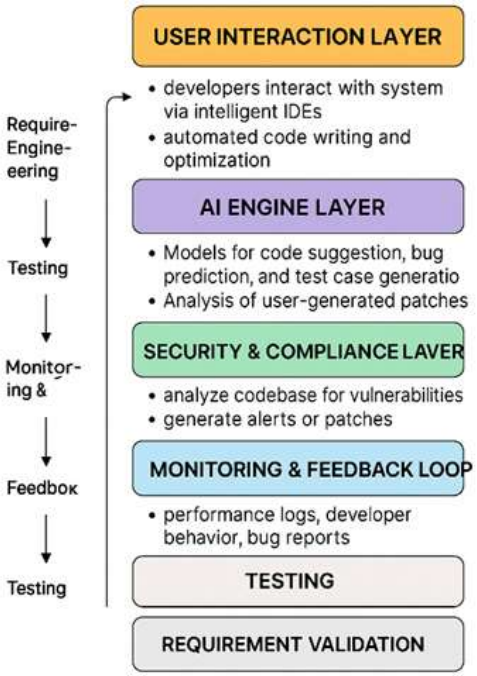

Figure 3: Proposed architecture of AI DevGuard

B. AI-driven software operations

The AI-driven development process described in the literature starts with activating a project environment, followed by user registration and access to the system [13]. AI automates the following steps:

1. **User Authentication and System Authorization:** Machine learning analyzes behavioral data, log data, and contextual information like geolocation to authenticate users and flag users that should be monitored. This enables dynamic access to be granted and unauthorized access attempts to be predicted and stopped. This improves security for the entire system.
2. **Distribution of Content and Access Based on Role:** AI makes smart decisions about when and how to give certain materials, updates, or content to people. By looking at user profiles and how people use content, the system customizes package deliveries to maximize performance and improve user experience.

3. **Confirmation of Data and Input Surveillance:** AI systems maintain and defend the integrity of Data by employing historical log, validation rules, and code quality standards. Anomalies and poor Data inputs are intercepted in real time. Data is stored and processed after confirming that org and regulatory standards are met [18].

4. **Monitoring of Lagging Indicators:** AI analyzes patterns of usage, bottlenecks, and resource consumption to provide optimization feedback. AI suggests system performance optimizations to users to address repetitive constraints and declining performance [19].

5. **Incorporation of Feedback and Flexibility:** AI can gather and analyze feedback, logs, and user interaction and behavior patterns. With reinforcement learning, AI can dynamically propose user interface, system logic, or feature changes and adapt the application to user-imposed constraints [20].

Figure 4: Proposed AI-Driven Software Operation

Continuous Maintenance and Learning: AI agents perform self-learning via real time feedback with no supervision. This helps the system to adapt to new environments, user behaviors, and security situations while maintaining consistent reliability and performance over time [16].

C. Comparing AI Platforms for the DevGuard Framework

For the selection of the AI platforms, OpenAI Codex and Google Cloud AI Platform receives the highest evaluation for relevance to AI-DevGuard framework [17]. All of them have distinct automation features for code generation, testing, threat detection and feedback loop throughout the SDLC.

Table 3: Comparison of AI Platforms for DevGuard Framework

Criteria	OpenAI Codex	Google Cloud AI Platform
Code Generation	Trained on large-scale codebases	Offers AutoML with general-purpose models,
Integration	Easily integrates with IDEs (VS Code, JetBrains)	Offers APIs
Security Support	Learns from secure code patterns, supports vulnerability pattern detection	Depends on external tools for security integration
Scalability	Scalable via API with usage-based limits	Fully scalable on GCP
Customization	Fine-tuning possible with proprietary code samples	Supports custom model training

V. SURVEY ANALYSIS AND FINDINGS

A. Survey Overview

To see the sentiments people, have and the way software developers utilize Artificial Intelligence (AI), a survey was designed. This survey was distributed to various participants such as software developers, project managers, team leads, and software development students. Considering the limited scope and time of this study, it analyzes the responses of 35 participants. While this is a small number, having this many responses allowed me to collect significant and diverse perspectives from various software development positions. Future research increase the survey scope to include a different, larger population to further improve the representativeness of the findings. It has provided a detailed breakdown of the most prevalent AI tools, their usage frequency, and the responses to the questions in the survey results in Appendix B. The survey aimed to capture the perspectives regarding the AI usage in different stages of the SDLC to gauge the value, potential issues, and future expectations [21].

B. Key Questions from the Survey

1. What is your professional background and years of experience in software development?
2. Are you familiar with the use of AI in software development?
3. Which AI tools have you used in any phase of software development?
4. In which software development phase have you used AI tools the most?
5. To what extent do you agree with the statement: "AI has improved the efficiency of the software development process?"
6. What benefits have you observed while using AI in software development?
7. What are your concerns regarding the use of AI in this field?
8. Do you believe AI can replace human software developers in the near future?
9. What motivates you or your organization to adopt AI tools in the development lifecycle?

C. Survey Summary and Results

There was confirmation of the claim that all (100%) of the participants acknowledged and engaged with Artificial Intelligence (AI) technology. AI is very real and is being employed in software development. This makes sense. It shows that people, whether students or professionals, have integrated AI tools into their tech work.

Table 4. Survey results and summary

AI Tool	Number of Users
ChatGPT	27
GitHub Copilot	19
Test.AI	7
DeepCode	6
Amazon CodeWhisperer	3
TabNine	5
SiemensGPT	1
Deepseek	1
grok	1
windsurf	1

There were interesting findings while reviewing the use of AI in software development. People said their favorite tools were ChatGPT and GitHub Copilot. Tools such as Test.AI, DeepCode, and TabNine showed moderate AI usage in the software development lifecycle. A few respondents tried out even more experimental and obscure technologies like Deepseek, Grok, and Windsurf. This shows a willingness to use a wide range of AI tools, even those that are newer and not widely accepted in the industry [22].

D. AI Usage in Software Development Phases

The respondents used AI in:

- Requirement Gathering
- Design
- Code Generation
- Testing
- Debugging

D. AI Benefits

The benefits of AI provided included:

- More efficient Generation
- More effective Requirement Analysis
- Thorough Testing
- Cost Efficiency AI
- Debugging

E. Problems Regarding AI in Software

The respondents voiced the following:

- Protection of Information
- Dependence on Automation
- Variance in Quality of AI Code
- Job Automation
- Placement of Employee

F. Data Analysis

The research had an iterative and non-linear analysis, as aligned with the abductive approach. The data gathered from the literature, documented case studies, surveys, and technical sources were evaluated to determine the role of AI in software development, especially in requirement engineering, debugging, code generation, testing, and various other areas. Discrepancies were resolved through consultation and other means with the instructors on software development and AI research. Most people are happy with how AI is used in software development. Respondents mentioned that code generation speed, clarity of requirements, accuracy of testing, cost saving, and workload reduction were larger benefits [23]. Respondents also mentioned security, dependency on AI, whether AI accuracy, ethics, and job replacement are issues. The feedback from the survey was analyzed by separating the responses by perception type, whether positive or negative, regarding software development and the use of AI. The positive responses centered on benefits brought by the use of AI such as quicker code generation and analysis, more precise testing, cost savings, and less manual effort needed. For negative responses, the key issues are security and privacy, dependency on AI, accuracy of results, ethics, and job loss [24].

Most people said there were multiple reasons why AI was adopted, and they saw the big gains in

efficiency and improved accuracy. It also helped reduce total workload. There seems to be little concern around the adoption of AI since 20% of people raised issues such as security, overuse, and ethics. Some friction points in software development AI are distributed and can be positively addressed [25].

VI. CONCLUSION

The AI-DevGuard Framework was presented in this paper as a way to secure, automate, and intelligently improve the Software Development Life Cycle. The study showed how combining machine learning, natural language processing, and code analysis can greatly enhance continuous testing, debugging, code generation, and requirement validation. The findings also demonstrate that by automating repetitive tasks, AI tools improve early vulnerability detection, reduce human oversight errors, and shorten development timelines. All things considered, AIDevGuard creates a safe and organized automation pipeline that helps developers at every stage of the software development life cycle (SDLC) and provides a preliminary guide for combining AI-driven automation with human supervision. Future research should assess practical applications, develop ethical governance for responsible AI use in software engineering, and improve security features to counter evolving threats.

REFERENCES

Jain, P., Kumar, S., & Sharma, P. (2022). Role of artificial intelligence in modern software development lifecycle: A review. International Journal of Computer Applications, 184(24), 1–7

Sharma, A., & Saxena, A. (2021). Artificial Intelligence in Software Development: A Systematic Literature Review. Procedia Computer Science, 185, 247–254.

Bhardwaj, A., Choudhary, V., & Sain, M. (2022). AI-based automated software testing frameworks: A comprehensive review. Journal of Systems and Software, 183, 111114.

Morley, J., Floridi, L., Kinsey, L., & Elhalal, A. (2020). From what to how: An initial review of publicly available AI ethics tools, methods and research to translate principles into practices. Science and Engineering Ethics, 26(4), 2141–2168.

Zhou, H., Xie, T., & Lu, Y. (2022). Machine learning for software engineering: A survey. ACM Computing Surveys, 55(1), 1–38.

Tufano, M., Watson, C., Bavota, G., Poshyvanyk, D., White, M., & Hata, H. (2021). An empirical study on learning bug-fixing patches in the wild via neural machine translation.

Zhang, Y., Luo, X., & Yu, H. (2022). Enhancing software testing using artificial intelligence: A comprehensive survey. Software: Practice and Experience, 52(2), 311– 332.

Chen, M., Tworek, J., Jun, H., Yuan, Q., et al. (2021). Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374.

Johnson, T., & Verma, P. (2025). AIOps for Modern DevOps: Enhancing Software Deployment with Machine Learning. ACM Transactions on Software Engineering and Methodology, 34(2), 1–27.

Pandey, P., Agrawal, P., & Bhatt, R. (2022). Artificial intelligence in software architecture design: A survey and future directions. Journal of Systems and Software, 189, 111328.

Lee, S., & Ali, M. (2024). Advancements in AI-Powered Code Generation Tools: A Comparative Study of Codex and Copilot. IEEE Access, 12, 12345–12358.

Sharma, D., Gupta, R., & Shukla, R. (2021). AI-enabled software engineering: A comprehensive survey. Procedia Computer Science, 192, 3359–3368.

Ahmad, A., Alam, M., & Hussain, M. (2023). Role of AI in software engineering: Tools and trends. International Journal of Advanced Computer Science and Applications, 14(2), 88– 96.

K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for Android applications," in Proc. 25th Int. Symp. Softw. Testing Anal., 2016, pp. 94–105.

Zhang, Y., & Kumar, R. (2025). AI-Driven Requirement Engineering: A Systematic Review. Journal of Systems and Software, 198, 111234.

M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "A survey of machine learning for big code and naturalness," ACM Comput. Surv., vol. 51, no. 4, pp. 1–37, 2018.

Jiang, Y., Li, J., Zhang, X., Zhang, C., & Wang, T. (2021). A survey of machine learning for big code and naturalness. ACM Computing Surveys (CSUR), 54(3), 1–37.

K.-J. Stol and B. Fitzgerald, "A holistic overview of software engineering research strategies," J. Syst. Softw., vol. 109, pp. 63–79, 2015.

Alshuqayran, N., Ali, N., & Evans, R. (2021). A systematic mapping study of DevOps: Current state and future directions. Journal of Systems and Software, 172, 110736.

Xu, Y., Yu, L., Wang, Y., & Ma, Y. (2022). Applying NLP for software requirements engineering: Current trends and challenges. Journal of Systems and Software, 188, 111260.

Kaur, R., & Kaur, D. (2020). AI-based requirements engineering: A review. International Journal of Scientific & Technology Research, 9(4), 3193–3197.

Pandey, P., Agrawal, P., & Bhatt, R. (2022). Artificial intelligence in software architecture design: A survey and future directions. Journal of Systems and Software, 189, 111328

Pan, Y., Ren, Y., Chen, X., & Liu, T. (2023). Artificial intelligence in DevOps: Automation and beyond. ACM Computing Surveys, 55(9), 1–36

Miah, M. (2020). Blockchain technology in peer-to-peer elearning: Opportunities and challenges. In Proceedings of the EDSIG Conference ISSN (Vol. 2473, p. 4901).

Chen, M., Tworek, J., Jun, H., Yuan, Q., et al. (2021). Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374.