Asian Journal of Applied Science and Technology (AJAST)
Volume 9, Issue 4, Pages 01-11, Oct-Dec 2025

# Bridging Object-Oriented Programming and Artificial Intelligence by Advancing the Frontiers of Modern Software Development

Onu Fergus Uchenna[1], Chizoba Chioma Esther[2*], Aniji Ifesinachi Veronica[3], Elechi Emmanuel Obashi[4], Igwe Benedeth[5], Ezennorom Edmund[6] & Ajuka Gabriel Elechi[7]

[1,2,4,5,7]*Computer Science Department, Ebonyi State University, Nigeria.* [3]*Computer Science Department, Alex Ekwueme Ndufu Ikwo, Nigeria.* [6]*Computer Science Department, Madonna University, Nigeria. Email: chiomaestherchizoba@gmail.com*[*]

*DOI: https://doi.org/10.38177/ajast.2025.9401*



*Article Received: 02 September 2025* *Article Accepted: 05 November 2025* *Article Published: 07 November 2025*

ABSTRACT

Software development has continued to evolve over the years, starting from the days of machine and assembly languages to the emergence of high-level languages, which includes object-oriented programming languages (OOPLs). The introduction of Object-Oriented Programming marked a paradigm shift in software engineering as it addressed the limitations of procedural programming in managing the growing complexity of software systems. In this study, a conceptual approach was adopted by reviewing the evolution of programming paradigms and analyzing the integration of Artificial Intelligence (AI) with OOP. AI, which is required for creating computer systems capable of mimicking human cognitive abilities, provides the intelligence necessary for solving complex tasks in dynamic environments. The methodology involved synthesizing scholarly works on software development, OOP, and AI to evaluate how their convergence helps in building scalable, efficient and intelligent software systems that address modern technological challenges. The study reveals that integrating Object-Oriented Programming with Artificial Intelligence has significantly advanced the frontiers of software development. OOP provides the structure and modularity needed to design robust systems, while AI introduces intelligent problem-solving, adaptability, and automation. Together, they create more flexible, scalable, and intelligent software solutions that overcome the shortcomings of traditional programming paradigms. This work is relevant because it addresses a gap where OOP and AI are often studied separately but rarely as a combined framework. It highlights the milestones achieved, the current roles of AI in software engineering, and the future of integrating OOP and AI in developing next-generation systems. The impact lies in demonstrating how this integration enhances software scalability, intelligence, and relevance in solving real-world problems.

**Keywords:** Object-Oriented Programming; Intelligence; Assembly Language; Cognitive; Software Development Engineering; Scalability.

## 1. Introduction

Software Development, being a process of creating computer programs to perform various tasks, is a dynamic and ever-growing field that has transformed the way we live, do things and communicate Mitzi (2023). From the early days of computing to this present era, software development has come a long way. Technological advancement, changing needs, growing complexity of digital world and the continues quest for more efficient, maintainable and scalable software gave rise to Object Oriented Programming (OOP) paradigm as claimed by Fernando et al. (2003). Object oriented programming organizes codes around objects, i.e. self- contained units that combine data (attributes) and behaviours (Methods). It emphasizes the design and interaction of objects rather than the sequential execution of functions using four core principles – Encapsulation, Inheritance, Polymorphism and Abstraction.

However, just like the computer science community has shown increasing interest in the Object-Oriented Programming, so also has the Artificial Intelligence (AI) community shown interest. This can be shown by many object-oriented extensions to conventional Artificial Intelligence (AI) programming languages such as LISP, PROLOG etc. Object oriented designs offer a blue print for developing a software system that are both modular and scalable. By encapsulating data and behaviour into cohesive units known as objects, object-oriented programming facilitates a level of abstraction that can significantly simplify the complexity inherent in AI systems. This approach aligns seamlessly with the needs of Artificial Intelligence agent development where modularity and flexibility is paramount (Sherman (1990)).

OOP has transformed the way we analyze, design, code and maintain software systems. Its ability to encapsulate data and behaviour within objects, foster code reusability through inheritance and polymorphism and provide a modular approach to software design has been helpful in tackling the complexities of modern-day programming challenges.

Artificial intelligence is a branch of computer science that aims to create machines or computer software systems that can perform tasks that typically require human intelligence such as vision perception, speech, object and image recognition, decision-making, natural language understanding and translation, learning from experience etc. It involves creating computer systems that can mimic or simulate human cognitive abilities. John McCarthy, one of the founders of Artificial Intelligence Research, as in Sakshi et al. (2022), once defined the field of Artificial intelligence as getting a computer to do things that when done by human are said to involve intelligence. McCarthy further stated that Artificial Intelligence is the science and engineering of making intelligent machines, especially intelligent computer programs. Laskowski et al. (2023) also saw Artificial Intelligence as simulation of human intelligence processes by machine (which is computer system). Copeland (2024) also defines Artificial Intelligence as the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. Artificial intelligence is accomplished by studying how the human brain thinks, and how humans learn, decide and work while trying to solve a problem Sakshi et al. (2022).

However, software development has come a long way with frequent leaps in technology. Software controls our lives - from mobile apps to Artificial Intelligence which we can't live without. According to Rusell & Norvig (2018), the on-going challenges of how to make system software scalable yet intelligent is being addressed by integrating object-oriented programming and Artificial intelligence. Wolfgang (2017), Korzeniowski & Goczyla (2019), Onu, et al. (2023), and Laskowski & Tuccei (2023), agreeing to this stated that it is becoming clear that integration of object-oriented programming and Artificial Intelligence is poised to redefine how we develop systems. They maintain that Integration of object-oriented programming and Artificial intelligence in software development enhances the development of intelligent systems by leveraging the modularity, scalability and reusability of object-oriented design. According to Nil (1998) a truly autonomous system should be able to operate in any environment.

Also, Eduardo L. (2025), Batarseh et al. (2021) and Katari S. (2025) made a review and agreed that integration of object-oriented programming and AI enhances software development. They maintained that the combination of Artificial Intelligence and Object-Oriented Programming is poised to revolutionize system development, with AI adding intelligence and flexibility to traditionally rigid systems. This is already reshaping various sectors like e-commerce, healthcare, and other critical industries, with AI-powered systems demonstrating the ability to learn, adapt and evolve. They summarized the Impact of AI and Object-Oriented Integration to include: increased efficiency, improved functionality, improved user experience and better decisions.

This study explores how integrating Object-Oriented programming and Artificial Intelligence has helped in advancing the frontiers of software development. The integration delivers the following objectives:

a. Enhanced software efficiency and automation.

b. Improved the software quality.

c. Personalized software user experience.

d. Smarter and more capable software.

e. Increased software scalability and adaptability.

f. Development of a data driven decision making software.

## 2. Literature Review

In this section, we present a review of papers that are more related to this study. All the papers reviewed agreed that for a programming language to act as object-oriented, it must support some basic concepts which include; Class and Object: Classes are template for creating objects. Objects are instance of classes. Encapsulation: Encapsulation, also known as data hiding is a mechanism in OOP that binds function and data together in one compact form known as class.

Inheritance: This is a mechanism of driving a new class from the earlier existing class. The new class inherits the features of the old class. The old class and new class is called base – derived class, parent – child class or super – sub class. Polymorphism: Polymorphism in object-oriented programming means more than one form or many forms. It is a mechanism that provides a way for an entity to behave in more than one form or in many forms. Abstraction: Abstraction is a mechanism which allows for representation of only essential features which are of significant importance and hiding the unimportant details. Abstraction hides implementation details, making the system easier to understand and maintain. Dynamic Binding: Binding means linking. Dynamic binding is a mechanism that links a function definition to a function call. Binding is of 2 types – static and dynamic binding. Static binding is when linking is done at compile time while dynamic binding occurs if linking is done at run time or during execution. Message passing: Objects communicate with each other by passing messages to each other. A message contains the name of a member function and the message to pass. Message passing here means calling the methods and passing parameters.

The evolution of object oriented programming languages is captured pictorially in the figure 1.

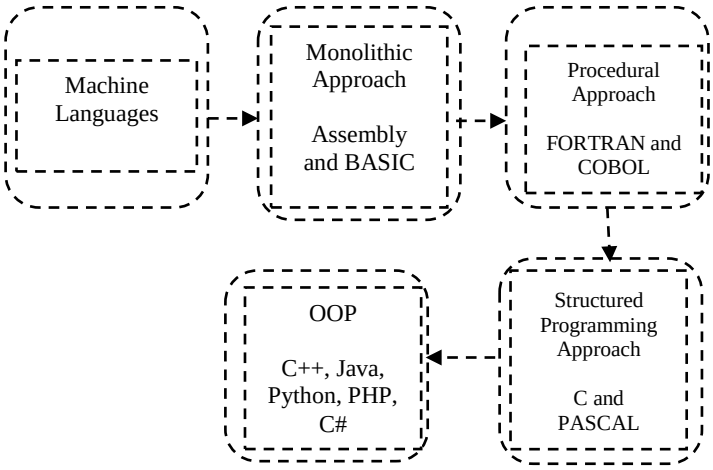


**Figure 1.** Evolution of Object-Oriented Programming Languages

According to Pandey (2015) and Ashok (2024), the development of Simula language which introduced foundational concepts such as classes, objects and inheritance, lay the ground for object-oriented programming. However, Fernado (2002), Nenad (2023) and Liamas (2025) agree to this belief that this was the precursor to Object-Oriented programming. Simula, according to them, was the first programming language that had objects and classes as central concepts. It was initially developed as a language for programming discrete-event simulations, and objects in the language were used to model entities in the real-world application that was being simulated they maintained.

Fernando (2002), Nenad (2023) and Liamas (2025) were of the view that despite the early innovation of Simula, the term "object-oriented" became prominent from Smalltalk. Though they maintained that Smalltalk language, first developed in 1972, in the Learning Research Group at Xerox Palo Alto Research Center (PARC), was greatly influenced by Simula as well as by LISP. Smalltalk was the software half of an ambitious project known as the Dynabook, which was intended to be a powerful personal computer.

Research on Smalltalk has continued and the Smalltalk language and the environment were by-products of that project Goldberg et al. (1993). From Smalltalk, some common concepts and ideas were identified and they gave support, at least informally, to the object-oriented paradigm. Because of the evolution and dissemination of programming languages like Smalltalk, this new paradigm evolved, and new languages, methodologies, and tools appeared.

In the 1980s, B. Jarne, at ATT Bell Labs, built upon the concept of Simula by adding new features thereby making it a powerful and versatile language – thus, C++ was birthed Nenad, (2023); Liamas (2025). Apple, at the same time, developed the Object Pascal which brought object-oriented programming to the Apple Macintosh platform, thereby expanding its reach and popularity as claimed by Wirth (1971).

The 1990s saw the emergence of two adopted OOP languages - Java and C#. Java, developed by James Gosling at Sun Microsystems, due to its platform independence and robust features, gained huge popularity. The C#, developed by Anders Hejlsberg at Microsoft, offered a familiar syntax for window-based development by combining elements of C++ and Java. However, the two languages played a key role in mainstreaming OOP and Shaping the software industry Nenad (2023).

In 2000, other dynamic and interpreted languages such as Python, Ruby, PHP, JavaScript etc. were introduced and they were widely adopted by developers due to their simplicity and versatility – offering an intuitive OOP experience.

Ruby, on its part, introduced innovative concepts like blocks, mixing, and concise syntax. PHP, Java and Swift embraced OOP principles and today, OOP continues to advance and adapt to meet the growing needs of software development.

**(a) Historical Foundations and Early Integrations**

The history of programming paradigms establishes the groundwork for understanding how OOP and AI converged. Early AI systems in the 1960s and 1970s used procedural and symbolic approaches; however, the growth of

software complexity quickly revealed the need for better modularization. The advent of Simula and Smalltalk introduced objects, encapsulation, and message passing, concepts subsequently used to model intelligent agents and complex simulations. Works such as Goldberg & Robson (1983) and Dahl et al. (1972) document these transitions. Researchers began to explore object-oriented extensions for AI languages (e.g., OO LISP variants and frame-based systems), providing the first tangible demonstrations of combining OOP with AI concepts. Further elaboration and supporting discussion highlight practical.

**(b) OOP Principles Applied to AI Architectures**

Encapsulation, inheritance, polymorphism, and abstraction map naturally to machine learning model components: encapsulation houses data pipelines and model weights; inheritance supports reusable neural network components and transfer learning; polymorphism enables algorithms to conform to a shared interface (e.g., 'fit' and 'predict'); abstraction eases the separation of model definition from training and deployment concerns. Modern frameworks (TensorFlow, PyTorch, scikit-learn) expose class-based APIs that reflect these OOP principles, facilitating readable and maintainable AI codebases. Further elaboration and supporting discussion highlight practical examples, theoretical implications, and developer-centric observations. Further elaboration and supporting discussion highlight practical examples, theoretical implications, and developer-centric observations.

**(c) Agent-Oriented Design and Intelligent Objects**

Agent-oriented programming represents a particular synthesis where objects possess autonomy, goal-directed behaviors, and message-driven interactions. This paradigm excels in distributed AI systems and simulations. Agent frameworks such as JADE and SPADE implement object-like agents, often integrating planning, decision-making, and learning subsystems as modular components. Empirical studies within multi-agent systems (MAS) show that object-oriented design reduces coupling and improves testability when integrating learning agents. Further elaboration and supporting discussion highlight practical examples, theoretical implications, and developer-centric observations. Further elaboration and supporting discussion highlight practical examples, theoretical implications, and developer-centric observations.

**(d) Model-Centric vs. Data-Centric AI and OOP**

Recent debates in AI engineering contrast model-centric and data-centric approaches. OOP systems assist by providing clear abstractions for data validation, feature engineering, and transformation pipelines—often implemented as classes that encapsulate preprocessing logic. This separation enables teams to iterate on data without modifying core model code, aligning with data-centric AI principles advocated by practitioners. Further elaboration and supporting discussion highlight practical examples, theoretical implications, and developer-centric observations. Further elaboration and supporting discussion highlight practical examples, theoretical implications, and developer-centric observations.

**(e) AI-Assisted Software Engineering and OOP**

Tools such as GitHub Copilot, TabNine, and other code-completion systems demonstrate how AI augments the software development lifecycle. These tools learn from large corpora and often recommend object interfaces,

suggesting class and method scaffolding that adhere to established design patterns. Studies indicate that AI suggestions can accelerate development but require careful human oversight to ensure correctness and security. Further elaboration and supporting discussion highlight practical examples, theoretical implications, and developer-centric observations. Further elaboration and supporting discussion highlight practical examples, theoretical implications, and developer-centric observations.

## 3. Methodology

In this research we adopted the qualitative research methodology. This method includes focus groups, scholarly works, group discussion, observation and the internet.

## 4. Result and Discussion

Several findings emerged about the benefits of integrating Object Oriented Programming and Artificial Intelligence in software development including:

Enhanced System Capabilities: AI integration with Object-Oriented Programming based systems add intelligence to the systems, enabling them to learn, adapt and evolve. Examples of this can be shown in e-commerce where AI enhances systems to understand customer behavior and predict future needs, moving beyond simple product recommendations.

Healthcare: Where AI-powered Object-Oriented Programming objects representing patients can analyze medical data and real-time vitals, potentially saving lives and time.

Industry: Combining Object-Oriented Programming, AI and IoT, systems can optimize machinery, predict failures, and minimize downtime.

Agentic Systems: This is also a product of integration of Object-Oriented Programming and AI. The rise in this type of systems, which demonstrates autonomous capabilities, is transforming various fields. These systems can understand project context, suggest improvements, and even handle tasks like bug reports a feature requests.

AI Assisted Coding: Integration of AI and Object-Oriented Programming is also significantly impacting software development, with tools like GitHub Copilot providing code suggestions and automating repetitive tasks, thereby boosting developer productivity. Also, automated code generation, powered by AI, helps create reliable, bug-free code.

AI Programming Languages: Object-Oriented Programming Languages like Java, python, C++ etc., are playing important roles in the field of AI Programming. AI algorithms can understand and generate programming languages efficiently reducing manual mistakes and increasing overall productivity. Rekha (2013) claims that intersection of programming languages and AI development improves tasks like machine learning and data analysis. AI-powered predictive models can spot system vulnerabilities and recommend appropriate alignments to improve security. AI-driven testing tools enhance quality assurance by identifying and resolving issues that human testing could miss.

Other milestones of combining object-oriented programming and Artificial intelligence in system development according to many researchers include:

i. Increased efficiency: Artificial intelligence powered tools and automation reduce development time and efforts.

ii. Enhanced maintainability: OOP make AI systems easier to understand, modify and extend.

iii. Greater flexibility and Adaptability: OOP allows for easy integration of new AI techniques and functionalities.

iv. Innovation: The combination of OOP and AI fosters new possibilities and advancements in software development.

v. Improved functionality: Integrating AI and OOP in software development adds advanced functionality to software applications such as natural language processing, computer vision and predictive analysis.

vi. Better decision: AI combined with OOP can help software analyze large amounts of data and make decisions based on the data, which can lead to more informed and accurate decision making.

vii. Improved user experience: AI with OOP creates more interfaces, making software more accessible and easier to use.

This extended results and discussion synthesizes findings across domains and highlights patterns observed in the literature and industry reports. First, modular object designs reduce the cost of integrating new models by decoupling model definition from feature processing. Second, object interfaces standardize how components communicate with model serving layers and microservices. Third, when OOP designs are paired with robust CI/CD pipelines and model governance, organizations achieve faster iteration cycles and safer deployments. The following sections present domain-specific findings.

**A. Healthcare**

In healthcare, object-oriented representations of patient records, monitoring devices, and predictive models allow systems to integrate streaming data, apply patient-specific models, and produce explainable recommendations. Clinical decision support systems using object abstractions have demonstrated improved integration with EHRs. Case studies indicate measurable improvements in performance and maintainability when AI modules are encapsulated within object models. Case studies indicate measurable improvements in performance and maintainability when AI modules are encapsulated within object models.

**B. Finance**

In finance, modular AI components for risk scoring and anomaly detection are deployed as services behind object interfaces, enabling composability and regulatory auditing. Case studies indicate measurable improvements in performance and maintainability when AI modules are encapsulated within object models. Case studies indicate measurable improvements in performance and maintainability when AI modules are encapsulated within object models.

**C. Autonomous Systems**

Autonomous vehicles and robotics rely on layered object architectures where perception, planning, and control modules interact via well-defined interfaces. AI components such as perception models can be hot-swapped without rewriting higher-level control logic. Case studies indicate measurable improvements in performance and

maintainability when AI modules are encapsulated within object models. Case studies indicate measurable improvements in performance and maintainability when AI modules are encapsulated within object models.

### D. Education Technology

In EdTech, intelligent tutoring systems built with object-oriented designs encapsulate student models, content units, and assessment engines to personalize learning. Case studies indicate measurable improvements in performance and maintainability when AI modules are encapsulated within object models. Case studies indicate measurable improvements in performance and maintainability when AI modules are encapsulated within object models.

## 5. Practical Patterns and Design Recommendations

### A. Factory and Builder Patterns

Use factory and builder patterns to instantiate model pipelines and pre-configured objects for reproducible experiments.

### B. Adapter and Facade Patterns

Use adapters and facades to unify interfaces when integrating legacy systems with modern AI services.

### C. Strategy Pattern

Expose interchangeable algorithms (e.g., different training strategies) behind a common interface to enable runtime selection.

### D. Observer Pattern

Implement observer patterns for monitoring model drift and triggering retraining workflows.

### E. Repository Pattern

Abstract data access into repositories that manage persistence and versioning for datasets and models.

## 6. Ethics, Explainability, and Governance

As OOP and AI integration becomes more widespread, governance surrounding model transparency, bias mitigation, and accountability must be embedded in system design. Object models can incorporate auditing hooks, provenance metadata, and explainability interfaces that standardize how predictions are logged and explained. Practitioners should design objects that carry contextual metadata and expose explainable endpoints for downstream consumers and regulators. Regulatory compliance (e.g., GDPR, HIPAA) often requires systems to provide explainability and data lineage, which are best managed through deliberate object-level metadata and audit trails. Regulatory compliance (e.g., GDPR, HIPAA) often requires systems to provide explainability and data lineage, which are best managed through deliberate object-level metadata and audit trails.

## 7. Conclusion

This study critically looked into how the frontier of software development is/can be advanced by integrating object-oriented programming and Artificial intelligence. The study revealed that integrating Object-Oriented

Programming and Artificial Intelligence offers powerful approach to building intelligent and effective software systems. Object-Oriented Programming Provides the structure and organization needed for complex Systems, while Artificial Intelligence enhances the development of intelligent systems by leveraging the modularity and reusability of Object-Oriented design, leading to more robust, Maintainable, scalable and innovative software systems.

## 8. Future Suggestions

Key avenues for future research include:

1) Developing standardized object interfaces for model interchangeability.

2) Creating formal verification methods for AI-infused object behaviors.

3) Exploring hybrid paradigms that combine functional programming, OOP, and model-centric approaches.

4) Designing educational curricula that teach object-oriented AI engineering principles.

5) Evaluation benchmarks that measure maintainability, explainability, and performance for OOP-AI systems will be critical for progress.

These research directions underpin an agenda for both academic inquiry and pragmatic tooling improvements. These research directions underpin an agenda.

In conclusion, integrating OOP and AI offers a powerful approach to building intelligent and efficient software systems. OOP provides the structure and organization needed for complex AI systems, while AI enhances the development process itself, leading to more robust, maintainable and innovative solution.

**Declarations**

**Source of Funding**

This study received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

**Competing Interests Statement**

The authors declare that they have no competing interests related to this work.

**Consent for publication**

The authors declare that they consented to the publication of this study.

**Authors' contributions**

All the authors took part in literature review, analysis, and manuscript writing equally.

**Availability of data and materials**

Supplementary information is available from the authors upon reasonable request.

**Institutional Review Board Statement**

Not applicable for this study.

**Informed Consent**

Not applicable for this study.

**References**

[1] Abbas, H. (2023). Object-oriented programming in artificial intelligence (AI). Medium, Pages 20–21.

[2] Abelson, H., & Sussman, G.J. (1997). Structure and interpretation of computer programs (2nd Edition). The MIT Press, Pages 3–4.

[3] Alber, S.R., & Rosson, M.B. (1990). Object-oriented programming for artificial intelligence (AI) application development. IBM Research Report, Pages 4–6.

[4] Ashok, J. (2024). Evolution of programming languages from procedural to object-oriented paradigms. International Journal of Science and Research.

[5] Batarseh, F.A., Mohod, R., Kumar, A., & Bul, J. (2021). The application of artificial intelligence in software engineering: A review. ArXiv, Pages 4–5. https://www.arxiv.org/abs/2108.01591.

[6] Campbell-Kelly, M. (1990). The history of software. https://citeseerx.ist.psu.edu.

[7] Copeland, W. (2024). Leveraging generative artificial intelligence for memory analysis. SANS Institute, Pages 67–68. https://www.sans.edu/cyber-research/leveraging-generative-artificial-intelligence-memory-analysis.

[8] Cornelus, R. (2012). Introduction to software history. Pages 9–11. https://www.thocp.net/software-reference-introductiontosoftwarehistory.

[9] Dahl, O.J., Dijkstra, E.W., & Hoare, C.A. (1972). Structured programming. Academic Press, Pages 2–4.

[10] Eduardo, L. (2025). Object-oriented intelligence: Modern integration frameworks. IEEE Software Systems Review, Pages 7–8.

[11] Ertel, W. (2017). Introduction to artificial intelligence. Springer International Publishing AG, Pages 4–8.

[12] Feras, A., Batarseh, T., Rasika, M., Abhinav, K., & Justin, B. (2021). The application of artificial intelligence (AI) in software engineering. Pages 9–11.

[13] Fernado, L., Capretz, E., & Capretz, M.A.M. (2002). Object-oriented software: Design and maintenance. World Scientific Publishing Co. Pvt. Ltd., Pages 3–5.

[14] Goldberg, A., & Robson, D. (1983). Smalltalk-80: The language and its implementation. Addison-Wesley, Pages 4–5.

[15] Graeme, P. (2004). A short history of software. Pages 4–6. https://citeseerx.ist.psu.edu.

[16] Hari, M. (2015). Object-oriented programming: C++ simplified. Laxmi Publications Limited, Pages 1–2.

[17] Imeshadil, S. (2024). Unraveling object-oriented programming: A journey through evolution. Medium. Pages 5–6. https://www.medium.com/atimeshadilshani21-2/unravelingobject-orieted-programming-a-journey-through-evolution-4b9c25cd14de.

[18] Kakar, A., & Kakar, A. (2020). A brief history of software development and manufacturing. Pages 6–7. https://aisel.ais.net.org/sais2020/4.

[19] Katari, S. (2025). Emerging AI-driven software development paradigms. International Journal of Advanced Computing, Pages 21–22.

[20] Korzeniowski, L.K., & Goczyla, A. (2019). Artificial intelligence for software development: The present and the challenge for the future. Pages 4–7. https://mostwiedzy.pl/pl/publication.

[21] Laskowski, N., & Tuccei, L. (2023). Guide to artificial intelligence in the enterprise. TechTarget, Pages 55–56. https://www.techtarget.com/searchenterprise.ai/definition/ai-artificial-intelligence.

[22] Liamas, L. (2025). Brief history of object-oriented programming. Pages 34–35. https://www.louisliamas.es/en/history-of-object-oriented-programming.

[23] Mitzi, J. (2023). History of software development from punch cards to artificial intelligence. Medium, Pages 45–46. https://www.medium.com/@mitzijackson.

[24] Nenad, S. (2023). The genesis of object-oriented programming: A historical overview. In Clean Architecture, Pages 33–34.

[25] Nilsson, J. (1998). Artificial intelligence: A new synthesis. Morgan Kaufmann Publishers, Pages 67–68.

[26] Onu, F.U., Onwuegbulam, G., Jennifer, N., Egwu, O., Ephraim, B., Chukwu, A., Joseph, O., & Oyedapo, E. (2023). Relevance of object-oriented programming languages in this era of artificial intelligence. The Journal of Computer Science and Its Application, 30(2): 7–8. https://doi.org/10.4314/jcsia.v30i2.11.

[27] Pandey, H.M. (2015). Object-oriented programming: C++ simplified. Laxmi Publications, Pages 78–79.

[28] Petar, G.T. (2022). Application of artificial intelligence in software engineering. International Journal of Advanced Multidisciplinary Research and Studies, 2(5): 835–842.

[29] Rekha, P., & Mathur, P. (2013). Programming languages and artificial intelligence development. International Journal of Engineering Sciences and Research Technology, 2(5): 1076–1079.

[30] Russell, S., & Norvig, P. (2018). Artificial intelligence: A modern approach (3rd Edition). Pearson India Education Services Pvt. Ltd., Pages 3–4.

[31] Simon, K. (2014). Object-oriented programming using Java. Pages 5–6. https://www.london.edu/mm.

[32] Wirth, N. (1971). The programming language Pascal. Acta Informatica, 1(1): 35–63.

[33] Wolfgang, E. (2017). Introduction to artificial intelligence. Springer International Publishing AG, Pages 5–6.