



Inteligencia artificial en la ingeniería de software

Artificial intelligence in software engineering

Samuel Salazar¹ 

¹ Universidad de Nariño, San Juan de Pasto, Colombia.

Submitted: 01-07-2026 | Revised: 04-09-2026 | Accepted: 07-10-2026 | Published: 08-10-2026

ABSTRACT

Generative Artificial Intelligence (GenAI) and Language Modeling (LLM) have taken on a significant role in software engineering, acting as a universal translator between natural language and technical specifications. These tools help optimize processes such as requirements engineering, test automation, and code generation, among others, allowing for valuable time savings in agile production environments. However, implementation can lead to problems such as the emergence of "hallucinations," increased code complexity, and cybersecurity risks that require human oversight. In education, AI assistants encourage competency-based learning, enabling students to develop their logic through prompt engineering and structured reasoning. The correct use of these technologies is essential for improving the resilience and maintainability of modern software.

Keywords: Artificial intelligence, Software Engineering, Prompt Engineering, cybersecurity, LLM, GenAI.

RESUMEN

La Inteligencia Artificial Generativa (IAGen) y los modelos de lenguaje (LLM) han tomado un rol importante en la ingeniería de software actuando como un traductor universal entre el lenguaje natural y las especificaciones técnicas, estas herramientas ayudan a optimizar procesos como la ingeniería de requisitos, automatización de pruebas, generación de código, entre otros; permitiendo reducir tiempo valioso en entornos de producción ágiles. Sin embargo, la implementación puede conllevar a problemas como la aparición de "alucinaciones", aumento de complejidad en el código y riesgos de ciberseguridad que necesitan de supervisión humana. En la educación, los asistentes de IA incentivan un aprendizaje basado en competencias, permitiendo que los estudiantes evolucionen en su lógica mediante la ingeniería de prompts y el razonamiento estructurado; el uso correcto de estas tecnologías es primordial para mejorar la resiliencia y mantenibilidad del software moderno.

Palabras clave: Inteligencia Artificial, Ingeniería de software, Ingeniería de prompts, Ciberseguridad, LLM, GenAI.

INTRODUCTION

The integration of large language models (LLMs) and generative artificial intelligence (GenAI) has driven a significant transformation in software engineering, making it possible to reduce the gap between human intent expressed in natural language and technical specifications, as they function as a universal translator, facilitating the transformation of unstructured inputs into actionable requirements, improving communication, and optimizing the development life cycle.

This study describes the state of the art in artificial intelligence applied to software engineering. It analyzes how the use of AI assistants allows for a transition from passive instruction models toward interactive collaboration environments that enhance critical competencies.

To delve deeper into this analysis, this document poses 6 questions that seek to explore the impact of these tools on productivity, mental workload, and the quality of the generated code.

Regarding its organization, the document is structured into the following sections: Methods and Research Questions, where the research approach is detailed; Results, which present the findings on the use of AI in technical education; and Conclusions, where best practices for training future software developers are synthesized.

METHODS

To conduct the systematic literature review presented in this article, the Software Engineering research process approach was used. This methodological approach was defined by Barbara Kitchenham and allows for the structured identification, evaluation, and synthesis of the available evidence on a research topic. The main objective was to analyze the impact of artificial intelligence on different cycles of software engineering, considering aspects such as requirements engineering, software maintainability, cybersecurity, competency development, and prompt engineering.

Research Questions

The objective of this study was to present the results of a systematic review of scientific literature related to the use of Artificial Intelligence in software development. For the construction of the state of the art, the following questions were formulated:

- RQ1: How do language models impact the quality and management of software requirements?
- RQ2: What effects does generative AI have on the technical debt and maintainability of the software developed?
- RQ3: How does explainable AI contribute to the resilience and automation of software testing?
- RQ4: What strategies guarantee cybersecurity and privacy in software development using AI?
- RQ5: How does the use of AI assistants influence learning and competency development?
- RQ6: How does prompt engineering optimize secure code generation in resource-constrained environments?

To formulate these questions, the PICO model with its PIPOH variant was taken into account[2]; the concepts are detailed in Table 1.

Search criteria

The search strategy consisted of exploring the general terms in specialized databases.

Table 1. Definition of the general concepts with PIPOH.

Criterion	Description
Population	Artificial intelligence in software engineering
Intervention	Large language models (LLM), generative AI, and prompt engineering techniques.
Results	Publications of articles citing studies of artificial intelligence implemented in software engineering
Professionals	Software engineers, systems engineers
Context	Agile environments, resource-constrained systems, and educational environments

Source: Own elaboration.

In the selected databases, search terms combined with synonyms were applied to cover a greater number of documents to be evaluated. Table 2 presents the terms with their respective synonyms and filters.

Table 3 is based on fields such as title, keywords, abstract, and full document.

Table 4 shows the resulting search strings for each term, along with their synonyms and their corresponding filters.

Table 2. Term, synonyms, and filters to compose the search string.

Term	Synonyms	Additional filters
AI in software engineering	AI in software development	Software Engineering Software Industry Open Publication year >= 2021
	Intelligent systems in software engineering	
	AI-driven development processes	
	AI-assisted coding	

Source: Own elaboration.

Table 3. Main term and database.

Term	Databases
AI in Software Engineering	Redalyc
	IEEE Xplorer>
	DOAJ
	Science Direct

Source: Own elaboration.

Table 4. Search string.

Main term	Search string
AI in Software Engineering	("AI in software Engineering" OR "AI in Software Development" OR "Intelligent systems in software engineering" OR "AI-driven development processes" OR "AI-assisted coding") AND (Software OR Engineering Software Industry) AND (Publication year >= 2021) AND (open)

Source: Own elaboration.

Selection

For the selection process, the following items were taken into account: title and keywords. Likewise, inclusion and exclusion criteria were defined for the review.

Table 5. Dates of consultation and file downloads.

Database	Search date
Redalyc	23/02/2026
Research	24/02/2026
IEEE Xplorer	17/02/2026
Science Direct	18/02/2026
DOAJ	20/02/2026

Source: Own elaboration.

The inclusion criterion is: a scientific document related to studies of artificial intelligence applied to software engineering or its search synonyms.

Similarly, the exclusion criteria are: studies that do not contain search terms or synonyms, and documents that are not available for download. The selection of primary sources was carried out in 2 stages, each with the following phases: phase 1: elimination of duplicate articles; phase 2: elimination of non-downloadable articles; and phase 3: application of inclusion and exclusion criteria. Because 4 databases with their synonyms and filters were used in this systematic literature review, it was necessary to perform the queries on various dates, as shown in Table 5.

As a result of the database search, a total of 87 studies were found, and after applying the inclusion and exclusion criteria, a total of 29 documents were obtained.

Quality assessment

Data extraction and synthesis of results

The main objective of this study is to determine the state of the art regarding the use of artificial intelligence at different stages of software development; after the search for the terms was carried out in each database and the three phases were applied, the following findings were obtained.

Table 6. Quality assessment in the search and selection processes.

Main term	Search results	Duplicate files	Excluded files	Relevant files	Databases
AI in Software Engineering	4	0	4	0	Redalyc
	8	0	6	2	Research
	27	2	17	9	IEEE Xplorer
	30	2	18	10	Science Direct
	33	0	26	6	DOAJ
Total	102	4	71	27	

Source: Own elaboration.

Table 7. Studies included in the systematic review.

Year	Quantity	Referencias
2021	2	[3], [4]
2022	0	
2023	1	[5]
2024	7	[6], [7], [8], [9], [10], [11]
2025	15	[6], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26]
2026	4	[27], [28], [29]
Total	27	

Source: Own elaboration.

RESULTS

The results obtained from the systematic literature review process are presented below, organized into 6 sections addressing the use of artificial intelligence in different phases of software development. Impact of language models on the quality and management of software requirements.

The integration of language models (LLM) and generative artificial intelligence has driven a significant transformation in requirements engineering, acting as a universal translator that narrows the gap between human intent expressed in natural language and operational technical specifications. In the initial stages of capture and elicitation, the use of natural language processing (PLN) tools and conversational assistants enables the transformation of unstructured inputs from stakeholders into actionable requirements, facilitating the identification of key entities and the resolution of communication conflicts in real time[12],[13].

This conversational capability not only democratizes the development process by enabling non-technical users to participate through the generation of domain-specific languages (DSL), but also optimizes the software lifecycle by significantly reducing requirements production times in agile environments[14],[26].

With regard to quality assurance, methodological advances have enabled a shift from simple lexical analyses to deep semantic evaluations that address dimensions such as clarity, consistency, and completeness[15].

The application of vision and language models (VLM) represents a crucial innovation in this field, as it enables the automatic detection of visual defects in flowcharts and traffic scenarios, identifying local inconsistencies and spatial incompleteness that previously required error-prone manual reviews[12],[16]. Furthermore, the systematic alignment of user stories with standardized quality frameworks such as the INVEST model has been enhanced by the generative capability of LLM, allowing not only the detection of problems but also the suggestion of precise reformulations and the generation of acceptance criteria by simulating collaboration among roles[15].

The management and organization of large volumes of technical information also benefit from automation through scalable architectures; models based on architectures such as BERT have demonstrated high accuracy in the automatic classification of functional and non-functional requirements, surpassing traditional techniques by capturing contextual nuances and ambiguous phrases[12],[17]. Complementarily, the use of fuzzy logic and machine learning algorithms in prioritization tasks helps to manage the uncertainty and volatility inherent to dynamic projects, enabling constant reprioritizations without the prohibitive computational costs of manual methods[6].

Finally, language models contribute to traceability and systemic consistency by automating the mapping between various development artifacts, such as requirements, source code, and test plans[18]; the use of generative AI allows comparison of different views of the system—for example, the developer's code perspective versus the functionality perceived by the user—to identify semantic mismatches and ensure that all actors share a coherent understanding of the product[19]. However, the adoption of these technologies faces critical challenges related to model hallucinations, the lack of transparency in their decision-making processes, and the need to integrate specific technical knowledge of industry standards; these challenges underscore the importance of maintaining rigorous human oversight and formal validation processes[6],[12],[16],[17].

Effects of generative AI on technical debt and maintainability of developed software.

The implementation of generative artificial intelligence (AGI) in software development has an ambivalent influence on technical debt and maintainability, establishing a tension between the streamlining of tasks and the increase in structural complexity. On the one hand, analyses of the use of language models in software engineering projects indicate that the final code integrated in repositories tends to have a higher cyclomatic complexity, length and Halstead effort compared to the initial suggestions of AI, suggesting that human adaptation of these products tends to increase the logical density of the system[20].

This structural complexity is aggravated by the persistence of algorithmic errors that do not disappear with large language models (LLM), but instead are transformed into new forms of opacity dependent on data and prompt sensitivity, which demands a robust theoretical foundation to guarantee the reproducibility and validity of the[7]software.

Likewise, the presence of “hallucinations” or logically inconsistent results creates distrust in professional environments, compelling developers to perform exhaustive manual validation to prevent the introduction of vulnerabilities or failures that would compromise the future stability of the code[3]. Moreover, there is a latent risk of overreliance that could erode programmers' technical skills, hindering independent corrective maintenance in the long term as in-depth knowledge of system logic is diluted[8].

Nevertheless, GenAI and advanced techniques such as fuzzy logic offer advantages for managing volatile requirements and facilitating dynamic reprioritization, which is critical for avoiding software degradation caused by constant changes in scope[6]. In global development contexts, maintainability continues to depend on the standardization of documentation to reduce inconsistencies, a practice that can be reinforced by AI tools to improve communication in distributed teams[3]. Finally, the sustainability of AI-based platforms can be strengthened through the use of permissive open-source licenses that foster a community capable of contributing updates and new functionalities collaboratively[21].

Contribution of explainable AI to the resilience and automation of software testing.

First, explainability is essential to mitigate “algorithmic error,” which arises when there is a mismatch between the software's capabilities and the system requirements; methodological transparency allows developers to understand why a text analysis or code-generation tool makes

certain decisions, which facilitates the replicability of tests and the validity of results[7].

In the field of automatic code generation, the use of transparency tools, such as sequence salience maps (salience maps), helps users trust the results generated by language models (LLM) by making the tool's output process visible; open and transparent AI is essential in critical environments to ensure accountability and mitigate biases[8],[22].

Resilience in software testing is enhanced by AI-driven automation tools that include "self-healing" capabilities; tools such as Testim allow test scripts to automatically adjust to changes in source code, maintaining the reliability of continuous testing without constant human intervention[22]. Furthermore, the use of fuzzy logic techniques contributes to resilience by enabling systems to manage uncertainty and ambiguity in dynamic environments, adapting to frequent changes in project requirements[6]. On the other hand, the application of reference architectures and software forensics techniques enables the systematic evaluation and comparison of the integrity of complex systems, such as agent-based systems, reinforcing their structural stability[9].

Generative AI has revolutionized the creation of test cases and the identification of faults, significantly reducing time to market (time-to-market). The use of machine learning algorithms for predictive analytics also makes it possible to identify bottlenecks and prioritize error detection in critical areas of high complexity[22]; these algorithms have demonstrated superior capabilities in generating unit tests and in automated program repair (APR), and can correct errors in both human-written and machine-written code[20].

Strategies for guaranteeing cybersecurity and privacy in software development using AI.

To guarantee security, it is necessary to implement a multidimensional approach that spans system architecture, data management, and algorithmic validation; a fundamental strategy is to adopt the "local-first software" paradigm, which allows researchers and developers to maintain ownership and full control over their information by processing it directly in the browser or on local machines, minimizing risks associated with storage in third-party infrastructures or cloud servers[21]. Additionally, in critical sectors such as open banking, advanced data encryption techniques, cloud servers with high security standards, and access control systems based on user consent must be employed[22].

To mitigate privacy and copyright risks, it is crucial to configure AI tools to block the generation of code based on protected public sources; users must rigorously validate the generated code to verify the existence of omitted licenses and avoid ethical or legal infringements[8]; likewise, the use of Explainable Artificial Intelligence (XAI) is recommended to understand the logic behind model decisions, facilitating the audit of potential security failures[22].

The implementation of advanced prompt engineering makes it possible to refine the instructions given to AI, ensuring that the resulting code is more accurate and aligned with the technical requirements[20]. To combat "hallucinations" (false information generated by AI), external verification tools, input validation, and sequence saliency maps that increase the transparency of the results should be used. In addition, the use of strongly typed languages (such as TypeScript) in the development of AI interfaces helps to reduce the occurrence of logical errors and vulnerabilities[8],[21].

Development must align with models such as the Agent Systems Reference Architecture (ASRA), which defines specific components for the safety and survivability of the system[9]. It is essential to integrate AI-driven automated software testing that spans the entire software development life cycle (SDLC), including the use of "safety guardrails" based on natural language processing models to protect and supervise the outputs of AI models[22].

Finally, the selection of any AI analysis tool must be grounded in theories that ensure the validity of the constructs, avoiding algorithmic errors that compromise the integrity of the system[7].

Influence of the use of AI assistants on learning and competency development.

The integration of artificial intelligence (AI) assistants in the educational and professional environment has transformed learning dynamics, enabling a transition from passive instruction models to interactive collaboration environments that enhance the development of critical competencies. In complex disciplines such as medicine, the use of frameworks such as "Vibe Coding" allows educators to assume the role of clinician-developers, creating tools that simulate expert reasoning and the cognitive flow of decision-making[23],[24].

This approach makes it possible to close the gap between static theoretical knowledge and clinical practice through interactive simulations that help students internalize complex decision-making rhythms; moreover, these customized applications foster competency-based learning, making it easier for students to address specific knowledge gaps in a self-directed manner[24],[27].

The impact of these tools on productivity and mental workload is significant, although it varies according to the user's experience level; there is a moderating effect linked to seniority, while novice programmers depend critically on the clarity of explanations and the semantic support of AI to succeed in their tasks, whereas senior-level professionals use their own mental model to compensate for the conciseness of the assistants, achieving similar results with different tools[23].

The development of "prompt engineering" has emerged as a fundamental competency in this new era, allowing users to strategically guide models toward precise results aligned with project objectives. Throughout the training processes, a notable improvement is observed in the quality of the instructions generated by the students, who evolve from exploratory and imprecise queries toward the use of strategic constraints and specific contexts[10],[20].

Modular systems such as TIPS (Template-Integrated Prompting System) demonstrate that structured reasoning, through techniques such as "Chain-of-Thought" (chain of thought), improves the logical consistency and semantic depth of results, especially in environments with limited computational resources[23]; therefore, AI assistants increase efficiency in quality assurance by predicting modules prone to defects and automating the generation of complex test cases that could be omitted by humans[19].

In the field of software engineering, AI influences the development life cycle by creating new specialized roles, such as AI integration specialists and code optimizers[25]; likewise, tools such as Cypress Copilot accelerate the onboarding of new team members by providing structured code examples that reduce the learning curve and standardize development practices[10]; however, AI should act as a complement and not as a substitute for human judgment, especially in strategic planning tasks or in the face of risks of biases and instruction injection attacks[19],[23].

Finally, the use of these assistants promotes a cultural shift toward continuous education within organizations, where technical experts ensure the digital literacy of the rest of the staff[4]. Although AI systems can make corrections at the semantic level and improve the readability of the work, human supervision remains indispensable to guarantee the safety and technical validity of the final results[19],[23].

Optimize prompt engineering for secure code generation in resource-constrained environments.

Prompt engineering optimizes code security by providing extensive and detailed instructions on syntax rules and domain constraints. For improved accuracy, it is recommended to include precise descriptions of the context and the task, examples of valid models, and a list of common errors identified empirically in the instructions; additionally, feedback cycles are conducted for error correction[14].

Regarding secure code generation, it is noted that models can learn insecure coding patterns from their training data, so prompt engineering should guide the assistant to actively apply input sanitization and restriction[28]. Without clear semantic guidance and a solid theoretical foundation in the design of instructions, the risk of "imperfections" increases, which manifest as errors in the source code or a misalignment between the user's intention and the generated output[7],[11]. Therefore, strategies such as few-shot prompting (providing specific examples) and Chain-of-Thought (step-by-step reasoning) are essential for the model to identify attack vectors and improve the structural quality of the solution[11],[12].

For resource-limited environments, such as mobile devices, optimization focuses on efficient energy and processing management. Since large models require considerable computational capacity, the use of cloud-based services over local solutions is favored[12]; in addition, algorithms have been proposed to optimize the scheduling of learning tasks on battery-powered devices, ensuring that AI inference does not compromise hardware autonomy[5]. In these contexts, the design of effective prompts allows AI to act as a "design copilot" capable of offering fast and accurate recommendations, reducing the developer's cognitive load and accelerating maintenance and refactoring processes without the need for heavy local infrastructures[12],[29].

CONCLUSIONS

Conducting a systematic literature review in the field of AI-assisted software engineering is essential due to the ambiguous nature of these technologies. Although language models (LLM) act as a universal translator that optimizes requirements capture and code generation, their implementation faces critical challenges such as "hallucinations" and a lack of transparency in their decision-making processes. Therefore, the importance of the review lies in the need to maintain

rigorous human oversight that mitigates algorithmic errors and validates semantic consistency, ensuring that the final product meets standards and does not introduce security vulnerabilities.

Within the framework of this study, the review of the state of the art has made it possible to identify how the use of AI assistants transforms technical education into a competency-based learning model. The importance of this analysis lies in showing that tools such as “Vibe Coding” enable teachers to simulate expert reasoning, helping students close knowledge gaps in a self-directed manner; moreover, this examination makes it possible to understand that success in implementing these tools depends not only on technology, but also on the user’s ability to counterbalance the assistants’ synthesis with their own mental model, thus avoiding the weakening of technical skills due to overdependence.

Finally, the six research questions posed allow us to conclude that generative AI significantly optimizes productivity and software resilience through “self-repair” capabilities in testing; however, a persistent tension is confirmed, as generated code can increase cyclomatic complexity and technical debt if there is no manual intervention. Therefore, prompt engineering and techniques such as “Chain-of-Thought” are key competencies for guaranteeing accurate and secure results, allowing future programmers to evolve from imprecise queries toward structured technical instructions that protect system integrity.

REFERENCES

1. B. Kitchenham, «Procedures for Performing Systematic Reviews», Keele, UK, Keele Univ., vol. 33, ago. 2004.
2. J. D. Martínez Díaz, V. Ortega Chacón, y F. J. Muñoz Ronda, «El diseño de preguntas clínicas en la práctica basada en la evidencia: modelos de formulación», *Enfermería Global*, vol. 15, n.o 43, pp. 431-438, jul. 2016.
3. H. U. Khan, M. Niazi, M. El-Attar, N. Ikram, S. U. Khan, y A. Q. Gill, «Empirical Investigation of Critical Requirements Engineering Practices for Global Software Development», *IEEE Access*, vol. 9, pp. 93593-93613, 2021, doi: 10.1109/ACCESS.2021.3092679.
4. D. Herremans, «aiSTROM—A Roadmap for Developing a Successful AI Strategy», *IEEE Access*, vol. 9, pp. 155826-155838, 2021, doi: 10.1109/ACCESS.2021.3127548.
5. L. M. Amugongo, A. Kriebitz, A. Boch, y C. Lütge, «Operationalising AI ethics through the agile software development lifecycle: a case study of AI-enabled mobile health applications», *AI Ethics*, vol. 5, n.o 1, pp. 227-244, feb. 2025, doi: 10.1007/s43681-023-00331-3.
6. N. Tasneem, H. B. Zulzalil, y S. Hassan, «Enhancing Agile Software Development: A Systematic Literature Review of Requirement Prioritization and Reprioritization Techniques», *IEEE Access*, vol. 13, pp. 32993-33034, 2025, doi: 10.1109/ACCESS.2025.3539357.
7. J. Wobst, «Avoiding algorithm errors in textual analysis: A guide to selecting software, and a research agenda toward generative artificial intelligence», 2025.
8. M. Basha, «Trust, transparency, and adoption in generative AI for software engineering: Insights from Twitter discourse», *Information and Software Technology*, 2025.
9. W. C. Regli et al., «Development and Specification of a Reference Architecture for Agent-Based Systems», *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 44, n.o 2, pp. 146-161, feb. 2014, doi: 10.1109/TSMCC.2013.2263132.
10. S. B. Nettur, S. Karpurapu, U. Nettur, y L. S. Gajja, «Cypress Copilot: Development of an AI Assistant for Boosting Productivity and Transforming Web Application Testing», *IEEE Access*, vol. 13, pp. 3215-3229, 2025, doi: 10.1109/ACCESS.2024.3521407.
11. J. D. Weisz, J. He, M. Muller, G. Hoefer, R. Miles, y W. Geyer, «Design Principles for Generative AI Applications», en *Proceedings of the CHI Conference on Human Factors in Computing Systems*, Honolulu HI USA: ACM, may 2024, pp. 1-22. doi: 10.1145/3613904.3642466.
12. M. Esposito et al., «Generative AI for software architecture. Applications, challenges, and future directions», *Journal of Systems and Software*, vol. 231, p. 112607, ene. 2026, doi: 10.1016/j.jss.2025.112607.
13. Md. A. Haque, «LLMs: A game-changer for software engineers?», *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 5, n.o 1, p. 100204, mar. 2025, doi: 10.1016/j.tbench.2025.100204.
14. N. Malamas, E. Tsardoulas, K. Panayiotou, y A. L. Symeonidis, «Toward efficient vibe coding: An LLM-based agent for low-code software development», *Journal of Computer Languages*, vol. 85, p. 101367, nov. 2025, doi: 10.1016/j.cola.2025.101367.
15. E. Wolf, A. Trendowicz, y J. Siebert, «Quality assessment of software requirements using artificial intelligence methods: A systematic literature review», *Information and Software Technology*, vol. 191, p. 107979, mar. 2026, doi: 10.1016/j.infsof.2025.107979.
16. N. Bukhary et al., «Few-Shot Evaluation of Vision Language Models for Detecting Visual Defects in Autonomous Vehicle Software Requirement Specifications», *IEEE Access*, vol. 13, pp. 117914-117942, 2025, doi: 10.1109/ACCESS.2025.3586554.
17. H. Ali, «Cloud-based machine learning for scalable classification of software requirements: Insights from the PROMISE dataset», 2025.
18. U. K. Durrani, M. Akpınar, H. Bektas, y M. Saleh, «Impact of Artificial Intelligence on Software Engineering Phases and Activities (2013–2024): A Quantitative Analysis Using Zero-Truncated Poisson Model».
19. J. R. Laracy, Z. Meng, V. D. Kirova, C. S. Ku, y T. J. Marlowe, «Software Quality Assurance and AI: A Systems-Theoretic Approach to Reliability, Safety, and Security», *Software*, vol. 4, n.o 4, p. 30, nov. 2025, doi: 10.3390/software4040030.
20. A. Naman, R. Shariffdeen, G. Wang, S. Rasnayaka, y G. N. Iyer, «Analysis of Student-LLM Interaction in a Software Engineering Project», 3 de febrero de 2025, arXiv: arXiv:2502.01273. doi: 10.48550/arXiv.2502.01273.
21. A. P. Placitelli, S. Hochscheid, F. Maffucci, A. F. Rees, G. Treglia, y A. Staiano, «Towards decentralized ecological monitoring: A local-first web-based AI platform for sea turtle detection», *Ecological Informatics*, vol. 93, p. 103569, feb. 2026, doi: 10.1016/j.ecoinf.2025.103569.
22. L. S. Ismail, A. Salim Jamil, A. Amantur Dastanbekovich, I. H. Mohammed AlDosari, y K. Salman, «AI-Driven Automation for Transforming the Future of Software Development», *تأاعال طا تییری دم و شزادرپ همان شهوژپ*, vol. 40, n.o 4, یسیلگنا همان هژیو 4, jul. 2025, doi: 10.22034/jipm.2025.728106.
23. W.-C. Chen y S.-Y. Chen, «TIPS: A Prompt Engineering Framework for Code Classification and Generation on Resource-Constrained Systems», *IEEE Access*, vol. 14, pp. 15723-15735, 2026, doi: 10.1109/ACCESS.2026.3658502.
24. F. Pesce y W. Cheungpasitporn, «Vibe Coding in nephrology education: clinician-led, AI-assisted development of open-source interactive learning tools», *Renal Failure*, vol. 47, n.o 1, p. 2581933, dic. 2025, doi: 10.1080/0886022X.2025.2581933.
25. N. Arshad, T. A. Butt, y M. Iqbal, «A Comprehensive Framework for Intelligent, Scalable, and Performance-Optimized Software Development», *IEEE Access*, vol. 13, pp. 74062-74077, 2025, doi: 10.1109/ACCESS.2025.3564139.
26. S. Sarferaz, «Implementing Agentic AI Into ERP Software», *IEEE Access*, vol. 13, pp. 178945-178960, 2025, doi: 10.1109/ACCESS.2025.3621887.

27. N. A. Mohd Salim, A. Azman, P. S. Sulaiman, N. S. Roslan, R. Hod, y S. K. Adam, «Development and validation of virtual patients for assessing history taking and communication skills among medical students», *BMC Med Educ*, vol. 26, n.o 1, p. 264, ene. 2026, doi: 10.1186/s12909-025-08503-x.
28. M. Glas, C. Nirschl, B. Lanyado, y J. Van Niekerk, «Insecure by design? A human-centric security perspective on AI-assisted software development», *Computers & Security*, vol. 164, p. 104842, may 2026, doi: 10.1016/j.cose.2026.104842.
29. F. Lonetti, A. Bertolino, P. Antonino, y D.-H. Bae, «Recent developments in software engineering for systems-of-systems and software ecosystems», *Journal of Systems and Software*, vol. 236, p. 112819, jun. 2026, doi: 10.1016/j.jss.2026.112819.

FUNDING

No financing

CONFLICT OF INTEREST

None

AUTHORSHIP CONTRIBUTIONS

Drafting – original draft: Samuel Salazar.

Writing–review and editing: Samuel Salazar.