

Switching Perspectives in the Era of Artificial Intelligence-Assisted Software Development

Muhammad Asri Bin Muhamad Khidzi*¹, Muhammad Faiz Bin Supian¹, Abdul Azim Bin Abdul Rashid², and Jack Febrian Rusdi²

¹Jabatan Kejuruteraan Perisian, Fakulti Teknologi Maklumat Dan Komunikasi, Universiti Teknikal Malaysia Melaka, Malaysia

²Jabatan Sistem Dan Komunikasi Komputer, Fakulti Teknologi Maklumat Dan Komunikasi, University Teknikal Malaysia Melaka, Malaysia

DOI: <https://doi.org/10.47772/IJRISS.2026.100800810>

Received: 04 September 2026; Accepted: 09 September 2026; Published: 18 September 2026

ABSTRACT

The development of generative AI has changed the way software is developed by enabling code and system components to be generated through natural language instructions. Approaches such as AI-assisted software development and vibe coding have the potential to reduce the effort required to produce software, particularly for routine programming tasks and prototype development. However, the ease of producing code does not necessarily reduce the need for human judgment; on the contrary, as code becomes easier to produce, individuals involved in software development need to evaluate the results from multiple perspectives to ensure that the functionality, user experience, structure, security, and behavior of the system are considered holistically. This study proposes Perspective Switching as a conceptual lens for understanding how individuals attend to different aspects of a software system and explores its initial empirical manifestation through measures of perspective coverage and priority. A quantitative exploratory study was conducted using a short questionnaire containing Likert-scale items, behavioral checklists, and preference questions, distributed to 29 respondents who had experience developing, testing, using, or evaluating software systems. Findings show that respondents consistently considered more than one perspective, with an average coverage of 3.7 of the 6 analytical domains (functional, user, robustness/unexpected-use, system, maintainability, security/misuse) per respondent. The robustness/unexpected-use and system perspectives were considered most frequently, while the security/misuse and maintainability perspectives were considered least frequently. This study provides initial evidence of the existence and variation of Perspective Switching and discusses its importance in the context of AI-assisted software development, where the human role shifts from code production to critical evaluation and validation.

Keywords: Perspective Switching; AI-Assisted Software Development; Vibe Coding; Software Evaluation; Multi-Perspective Cognition

INTRODUCTION

Traditionally, software development involves several key activities such as understanding requirements, designing solutions, writing code, testing the system and performing maintenance (Morgan & Jamison, 2025). In this process, the ability to produce code is often an important part of a developer's skills. This situation has led to the attention of software development sometimes being focused more on the question of whether a system can perform a specified function. However, software is used in more complex situations than simply performing a specific function. It interacts with users, data, other systems and the operating environment, in addition to having to deal with changing requirements, user errors and security threats (Oruthotaarachchi & Wijayanayake, 2025). Therefore, a system that successfully produces the desired output does not necessarily indicate that the system has good quality, because aspects such as usability, security, maintainability and durability also affect the overall quality of the software. The development of

generative AI has subsequently brought about more significant changes in the way software is produced. AI systems can now help generate code, suggest solutions, build components, explain code, correct errors and produce prototypes based on instructions in natural language(Shethiya, 2025).

This development has also led to approaches such as AI-assisted software development and vibe coding,

which allow individuals to generate a large portion of code by instructing an AI system and evaluating the results based on the behavior of the program produced (Kauppinen, 2026). This change has made the activity of generating code easier than ever before, especially for routine programming tasks or prototyping (Shethiya, 2025). However, the ease of generating code does not mean that the software development process has become less demanding of human judgment. On the contrary, as code becomes easier to produce, human attention increasingly needs to be paid to the questions of whether the code meets the requirements, how it will be used, and what problems may not be apparent in the initial results (Melo et al., 2022; Yoo & Kim, 2025).

This change indirectly changes the role of humans in the software development process. In situations where humans write most of the code themselves, the process of understanding the problem, generating solutions, and evaluating the results usually occurs in the same workspace. In AI-assisted development, humans can give instructions, receive the generated results, change the instructions, and re-evaluate the results repeatedly. In these situations, humans not only need to know what they want to build but also need to determine whether the results provided are truly acceptable (Gupta et al., 2023). This evaluation cannot necessarily be done by looking at just one thing. A function may produce the correct output but fail when receiving different inputs. A system may meet the needs of a particular user but be difficult for other users to use. Code may produce correct results but be difficult to understand and change in the future. Similarly, a system that appears to work well may have vulnerabilities when used in unexpected ways or when problems occur in other components. Several studies have also shown the risks of accepting AI-generated results without adequate evaluation (Gurunath Sakhalkar, 2025; Perry et al., 2023; Yoo & Kim, 2025). Among them, users assisted by AI models were found to produce less secure code than a control group, even though they themselves rated the code as more secure than the real situation. Another study that analyzed the code generated by GitHub Copilot also showed the existence of security vulnerabilities in a large part of the sample studied (Brasoveanu et al., 2020; Majdinasab et al., 2023).

This situation shows that the problems in software development are not only related to the ability to produce a solution, but also to the way a person views and evaluates that solution. When attention is only paid to the main functionality, other important aspects may not be considered. For example, an individual may view a system from a functional perspective and ensure that all instructions produce the correct output, but at the same time not think about the user experience, unexpected input conditions, possible abuse, code structure or the impact of a change on other parts of the system (Oruthotaarachchi & Wijayanayake, 2025; Tuomimaa, 2025). The same thing can happen when a person is so focused on a specific technical aspect that they ignore the context of the system's use. In other words, a system can look good when viewed from one angle but show weaknesses when viewed from a different angle. Therefore, the ability to shift attention from one aspect to another becomes increasingly important in the process of understanding and evaluating software.

In this context, this study focuses on the concept of Perspective Switching, which is the change of perspective when one thinks about a software system. Perspective Switching does not refer to an evaluation method that requires individuals to follow a specific sequence, but rather to the tendency to view a system from multiple perspectives when the situation requires it (Brasoveanu et al., 2020). For example, one might start by looking at whether the system functions correctly, then consider what happens when the user provides different input, how the system behaves when errors occur, whether the code is easy to maintain, or what the impact of changes on other parts of the system is (Melo et al., 2022). Such a shift in attention allows a system to be viewed in more than one way and has the potential to help identify things that are not visible when the evaluation is focused only on the main function.

As the use of AI in software development increases, this capability becomes increasingly relevant as humans increasingly play a role in reviewing, validating, and making decisions about the results produced by AI (Hu

et al., 2022). The question is no longer simply how well AI can produce code, but how humans think when faced with software produced or assisted by AI. Based on this question, this study explores how individuals with different backgrounds and roles consider various aspects when faced with a software system (Perry et al., 2023). The study focuses on the aspects considered, the behaviors performed, the priorities when making judgments, and the differences based on roles and experiences. This approach is expected to provide an initial picture of the existence and variations of Perspective Switching among individuals involved in the development, use, testing, or evaluation of software, as well as provide a basis for understanding its importance in the era of software development increasingly assisted by AI.

Conceptualization of Perspective Switching

Perspective Switching refers to a change in perspective when someone thinks about a software system, whether it occurs intentionally or in response to the circumstances encountered. The concept is based on the idea that a system can be viewed from more than one angle and that each angle can reveal different things. For example, when a system function produces the correct output, one can move on to questions about how the function is used by the user, what happens when the input changes, how the system behaves when errors occur, or whether the code structure allows the system to be changed in the future (Majdinasab et al., 2023; Roberts & Badi, 2026). Therefore, Perspective Switching in this study is not defined as a process that necessarily occurs consciously or in a specific sequence, but as the tendency of individuals to shift attention between different aspects of the software when thinking about, using, testing, or evaluating a system.

To enable this concept to be explored empirically, this study uses several categories of perspectives as the basis for initial analysis. These categories include the functional perspective, which looks at whether the functions and outputs meet the requirements, the user perspective, which looks at the experience and how the system is used, the robustness perspective, which looks at situations or usage that are different from the usual situations, the system perspective, which looks at the relationships between parts and components, the maintainability perspective, which deals with the structure and changes of the code, and the security perspective, which considers the possibility of system misuse (Kauppinen, 2026; Yilmaz & Karaoglan Yilmaz, 2023). These categories are not intended to be a definitive framework because the purpose of the study is to explore how individuals perceive a system. Instead, they are used as a starting point to identify patterns of judgment that emerge in the data. This approach is related to the study of perspective-based reading in software engineering, which shows that a software artifact can be examined from different role perspectives to increase coverage of the issues identified (Oruthotaarachchi & Wijayanayake, 2025; Rojas et al., 2022). This study takes this idea to a broader level by looking at the considerations of the software system.

The importance of Perspective Switching becomes even clearer in the context of AI-assisted development. When AI is used to generate code, humans increasingly play a role in giving instructions, reviewing results, validating the system's behavior, and determining whether the results are acceptable (Shethiya, 2025). In this situation, producing code is no longer the only challenge. Humans also need to understand things that may not be visible from the results generated. A system may fulfil the requested function but have problems when receiving certain inputs, be difficult to use by certain users, difficult to maintain or have vulnerabilities that can be abused (Gupta et al., 2023; Melo et al., 2022). Therefore, the value of Perspective Switching lies in the ability to shift attention from questions such as whether the system works to other questions related to how the system is used, how it can fail, how it is built and how it may affect other parts. In the context of AI-assisted development, this becomes increasingly important because humans need to evaluate the results produced more critically and not rely solely on the behavior of the system under normal circumstances (Hu et al., 2022; Perry et al., 2023).

Based on this conceptualization, the study focuses on two main questions. First, it identifies the perspectives that individuals most frequently consider when thinking about a software system. Second, it examines the extent to which individuals consider multiple perspectives when making judgments about software. Accordingly, the study addresses the following research questions:

RQ1: What perspectives are most frequently considered when individuals think about a software system?

RQ2: To what extent do individuals consider multiple perspectives when making judgments about software?

These research questions guided the design of the research instrument and the subsequent data collection and analysis.

RESEARCH METHODOLOGY

This study uses an exploratory quantitative design to explore how individuals consider different aspects when faced with a system or software. This study does not aim to test cause and effect relationships or determine the effectiveness of a software development method. Instead, the focus is on the respondents' patterns of consideration, the aspects they pay attention to, and the extent to which they view a system from more than one angle. This exploratory approach is in line with the initial nature of the study, which aims to obtain an empirical picture of the concept of Perspective Switching before the concept is tested with a more specific study design.

Data were obtained from 29 respondents who have experience in developing, testing, using or evaluating systems or software. Respondents were selected using convenience sampling based on their willingness to participate in the study. Respondents came from backgrounds of experience related to software development or use. However, the instrument used in this data collection did not record the respondent's role and level of experience in detail. Therefore, this study did not conduct formal comparisons between groups such as students, professional developers and evaluators. This information is proposed to be recorded in future studies to enable the Perspective Switching patterns to be compared based on the respondents' backgrounds and experiences.

The research instrument used a short questionnaire designed to take approximately two to three minutes to complete. The main part of the questionnaire consisted of six five-point Likert scale items that measured respondents' tendency to consider different usage situations, user experiences, errors, relationships between system parts, and code structure. In addition, respondents were asked to identify behaviors they had engaged in when dealing with a system and to select three aspects that most influenced their view of the system. This design allowed the study to obtain information not only about the respondents' level of agreement, but also about what they had done and the aspects that were their priorities when making decisions. If open-ended questions were also used in the instrument distributed to respondents, the responses were analyzed qualitatively to identify perspectives that emerged that were not guided by the list of options provided.

For perspective analysis, the nine behavioral options in the checklist were mapped to the six domains introduced in the study's conceptualization, namely functional, user, robustness or unexpected use, system, maintainability, and security or misuse perspectives, as shown in Table 1. Some items may represent the same domain because several different behaviors may indicate attention to similar aspects of the system. Based on the mapping, the perspective coverage score is calculated as the number of unique domains touched by the respondent's choices. This score ranges from zero to six and is used as an indication of the breadth of aspects considered by a respondent. This score is not intended to represent an absolute measure of Perspective Switching ability. Rather, it serves as an exploratory indicator of the breadth of perspectives represented in a respondent's reported considerations.

Table 1. Mapping of behavioral checklist items (Part B) to perspective domains

Item Code	Description of Part B Items	Domain Perspective
A	Trying different inputs from the given example	Robustness/Unexpected Use
B	Thinking about what happens when users do something unexpected	Robustness/Unexpected Use

C	Consider how the system is viewed/used by users	User
D	Checking how the system behaves when an error occurs	Robustness/Unexpected Use
E	Pay attention to the structure/organization of the code	Maintainability
F	Considering the relationships between parts of the system	System
G	Thinking about possible abuse of the system	Security/ Misuse
H	Comparing the system with the requirements/objectives	Functional
I	Thinking about system performance under different conditions	System

The data were then analyzed descriptively. Responses to the Likert-scale items were summarized using the mean, standard deviation, and percentage of agreement, with scores of 4 or 5 classified as agreement on the five-point scale. For the behavioral checklist, the frequency and percentage of selection of each item were calculated to identify the most common behaviors performed by respondents. The perspective coverage score was calculated as the number of unique domains represented in each respondent's selected behaviors. Preference questions were analyzed based on the frequency and position of aspects selected within the three main options. If open-ended questions were used, responses were categorized based on the identified perspective themes, including categories that were the same as the initial six domains as well as new categories if there were perspectives that were not expected from the initial framework. The results of this analysis were then used to answer two research questions regarding the most frequently considered perspectives and the extent to which respondents considered multiple perspectives.

RESEARCH FINDINGS

This study involved 29 respondents who responded to six Likert scale items related to considerations of a software system. As shown in Table 2, all items recorded mean values exceeding 4.00 on a five-point scale. Q1 and Q3 recorded the highest mean, 4.45 respectively. Q1 which related to considerations of system behavior in different situations also recorded the highest percentage of agreement, which was 100.0%. Q3 related to different user experiences and Q4 related to the effects of changes between parts of the system recorded a percentage of agreement of 96.6%. Q2 and Q6 recorded the same mean, which was 4.24, with percentages of agreement of 86.2% and 89.7% respectively. Q5 which related to the structure or way the system is built recorded the lowest mean, which was 4.10, with 86.2% of respondents giving a score of four or five. Descriptively, all items showed a high level of agreement among respondents.

Table 2. Descriptive statistics of Part A (N=29)

Item	Mean	SD	% Agree (score ≥ 4)
Q1 – Consideration of behavior in different situations even though the output is expected	4.45	0.51	100.0%
Q2 – Unexpected input or usage considerations	4.24	0.69	86.2%
Q3 – Different user experience considerations	4.45	0.57	96.6%
Q4 – Consideration of the impact of changes between system components	4.41	0.57	96.6%
Q5 – Consideration of system structure or construction other than function	4.10	0.62	86.2%
Q6 – Priorities change according to the aspects of the system being considered	4.24	0.64	89.7%

The analysis of Part B then looked at the behavior reported by respondents when dealing with a system or software. Based on the mapping of items to six perspective domains, the Robustness or Unexpected Use domain was the most frequently touched domain. All 29 respondents or 100.0% selected at least one item related to the domain. The System domain was touched by 26 respondents or 89.7%, while the User domain was touched by 21 respondents or 72.4%. The Functional domain was chosen by 13 respondents or 44.8%, followed by the Security or Abuse domain by 10 respondents or 34.5%. The Maintainability domain recorded the lowest number with eight respondents or 27.6%. Based on the six domains, each respondent had a coverage score between two and five domains. A total of 16 respondents or 55.2% touched four domains, eight respondents or 27.6% touched three domains, three respondents or 10.3% touched five domains and two respondents or 6.9% touched two domains. No respondent touched on only one domain or all six domains. Overall, the perspective coverage score recorded a mean of 3.69 with a standard deviation of 0.76.

Table 3. Perspective domain coverage based on Part B responses (N = 29)

Perspective Domain	Questionnaire Item(s)	Respondents	Percentage (%)
Robustness / Unexpected Use	A, B, D	29	100.0%
System	F, I	26	89.7%
User	C	21	72.4%
Functional	H	13	44.8%
Security / Misuse	G	10	34.5%
Maintainability	E	8	27.6%

Section C examines the aspects selected as the top three priorities by respondents. Output Accuracy was the aspect most frequently selected, by 22 respondents or 75.9%. User Experience was selected by 16 respondents or 55.2%, while Key System Functions were selected by 15 respondents or 51.7%. Error Handling was the next choice with nine respondents or 31.0%, followed by System Performance with eight respondents or 27.6%. Code Structure and Organization was selected by six respondents or 20.7%. Compliance with Requirements or Objectives and Relationships between System Components were selected by four respondents or 13.8% respectively. Handling Different Inputs was the aspect least selected, by three respondents or 10.3%.

Table 4. Frequency of highest priority aspects (Section C, N=29)

Priority Aspects	Number of Respondents	Percentage (%)
Output accuracy	22	75.9%
User experience	16	55.2%
Main functions of the system	15	51.7%
Error handling	9	31.0%
System performance	8	27.6%
Code structure and organization	6	20.7%
Compliance with requirements/objectives	4	13.8%
Relationships between system components	4	13.8%
Different input handling	3	10.3%

When the findings of Sections A, B and C are viewed together, there is a difference between the breadth of aspects reported to be considered and the aspects selected as priorities. Section A shows a high level of

agreement for the six items representing different system conditions and aspects. Section B shows that respondents touched on between two and five of the six specified domains, with an average of 3.69 domains per respondent. However, when asked to select the three aspects that most influenced their views, respondents' choices were more focused on Output Accuracy, User Experience and System Key Functions. This difference in patterns between considerations, domain coverage and priorities forms the basis for the study's discussion.

Coverage Patterns and Perspective Priorities

The findings of the study show that consideration of software is not focused on just one aspect. Based on Table 2, all six items show a high level of agreement, with means ranging from 4.10 to 4.45. However, the pattern obtained from Table 3 shows that the coverage of perspectives in reported behavior is not equal. Although the Robustness or Unexpected Use domain was considered by all respondents and the System domain by almost all respondents, the Security or Misuse and Maintainability domains were only selected by 34.5% and 27.6% of respondents, respectively. This shows that considering various aspects of software does not necessarily mean that each aspect receives equal attention.

A more significant difference can be seen when respondents are asked to select the three most important aspects. Output accuracy is the most dominant choice, followed by user experience and the main functions of the system. On the other hand, aspects such as handling different inputs, relationships between components, code structure and the possibility of system misuse receive lower choices. This pattern shows that there is a difference between the breadth of aspects that can be considered and the aspects that are ultimately given priority. In other words, an individual may recognize that several other aspects need to be considered, but when attention needs to be focused on a few key points, the assessment tends to revert to the accuracy of the output and the visible functionality. This distinction between coverage and priority is an important observation in understanding how individuals make judgments about software.

The lower ranking of the Security/Misuse and Maintainability domains also needs to be interpreted with caution. This finding does not mean that respondents ignored these two aspects, but rather that they appeared less frequently as reported behaviors or as priorities when compared to other domains. This is in line with the challenges reported in studies of AI-generated code, particularly where attention to functionality can lead to security issues not being fully assessed. In this context, the more important question is not simply whether individuals consider an aspect, but to what extent they are able to shift their attention to other aspects when evaluating a system (Brasoveanu et al., 2020; Yoo & Kim, 2025).

A Shift of Perspective in Ai-Assisted Development

In AI-assisted development, this question becomes increasingly relevant as the code generation process increasingly involves instructions, reviews and validation of the output produced by AI systems. This change makes the human ability to view software from different perspectives increasingly important. Code that appears to work well can still have weaknesses in terms of unexpected usage, user experience, interaction between components, security or maintainability. Therefore, the value of Perspective Switching in this context does not lie in the addition of another evaluation criterion, but in the ability to shift attention from one aspect to another when evaluating the development outcome.

This finding also has a connection to the tradition of perspective-based reading in software engineering, which uses a particular perspective to direct the evaluator's attention to different types of issues. The findings of this study can be seen as complementary to that tradition by showing that individuals already have a certain level of diversity of considerations before any perspective instructions are explicitly given. The coverage of between two and five domains observed in the study suggests that individuals already have a different level of diversity of considerations. This opens the door to more targeted interventions, such as training or checklists that help individuals identify domains that have not been considered when evaluating software, particularly in development situations involving AI output (Kauppinen, 2026; Majdinasab et al., 2023).

However, these findings do not prove that Perspective Switching improves the quality of evaluations or

produces better software. The measurements in this study reflect the breadth of consideration across domains rather than the perspective switches that occur dynamically throughout the evaluation process. In addition, the small sample size and sampling method used limit the generalizability of the findings. Further studies could build measurements that directly record the sequence of perspective switches and test whether perspective switch instructions can improve the detection of issues in software, including software produced with the assistance of AI.

RESEARCH LIMITATIONS

This study has several limitations in terms of sample size and selection. The study involved 29 respondents selected through convenience sampling, making the findings more appropriate to be interpreted as an initial picture of perspective consideration patterns and not generalizable to the broader software developer or user population. In addition, the study instrument did not explicitly record the role or level of experience of the respondents, such as the differences between student or junior developers, professional developers, and evaluators. Therefore, comparisons between groups based on role or experience could not be made in this study. Future studies could use a larger sample and systematically record the background of the respondents to enable more detailed comparative analysis.

Limitations also involve the instrument and scope of data collected. The current instrument only included Likert scale items, lists of behaviors, and preference choices, without open-ended questions that were originally designed to elicit additional qualitative explanations. Therefore, this study was unable to explore in depth the reasons why respondents paid attention to certain perspectives or why some perspectives became less of a priority. Further studies could incorporate open-ended questions or interview methods to understand the consideration processes underlying the observed quantitative patterns.

In addition, the mapping of the nine items in Section B to the six perspective domains was based on the researcher's conceptual considerations and has not undergone formal content validation, such as expert assessment or inter-rater reliability testing. Therefore, the perspective coverage measure used in this study should be viewed as an initial analytical indication rather than a fully validated measure. In terms of study design, the cross-sectional descriptive-exploratory approach does not allow for causal conclusions to be drawn regarding the impact of Perspective Switch on actual assessment quality. In particular, the measure used reflects the breadth of consideration across several domains rather than the sequence of perspective switches that occurs dynamically. Future studies could construct a formally validated instrument and use an experimental design to test whether perspective switch instructions can improve the ability to detect issues or assess software quality.

CONCLUSION AND CONTRIBUTION

This study explores the concept of Perspective Switching as a lens for understanding how individuals shift their attention between different aspects when thinking about, using, or evaluating a software system. Based on data from 29 respondents, the study provides initial empirical evidence that consideration of software is not limited to a single aspect. Respondents showed high levels of agreement on the consideration of multiple circumstances, user experience, changes between components, and system structure. However, the coverage of the observed perspectives was uneven. The Robustness/Unexpected Use and System perspectives appeared more frequently in reported behavior, while Security/Misuse and Maintainability were less frequently considered. At the same time, when asked to determine top priorities, respondents were more likely to focus on output accuracy, user experience, and the system's primary functions. This pattern suggests that the breadth of perspectives that can be considered is not necessarily the same as the aspects prioritized when making an evaluation.

This study makes three main contributions. First, conceptually, the study proposes Perspective Switching as a lens for examining how individuals distribute attention across different aspects of software evaluation. This concept complements previous perspective-based approaches by paying attention not only to the

perspective used for inspection, but also to how individuals allocate attention between different software domains. Second, from an empirical perspective, the study provides initial evidence on patterns of perspective coverage and preference among individuals with experience in software development, use, testing, or evaluation. The use of six perspective domains and a measure of domain coverage provide an initial form of operationalisation that can be refined and validated in future studies. Third, from a practical and current perspective, the study links the concept of Perspective Switching to the changing role of humans in AI-assisted software development. As AI is increasingly used to generate code and system components, the human ability to evaluate results from more than one perspective becomes increasingly important to avoid judgments that are too focused on the function or output that appears correct.

However, the findings of this study should be interpreted as preliminary evidence. The measures used in this study reflect the breadth of consideration across domains rather than the dynamic perspective switch that occurs. In addition, the small sample size, convenience sampling, and lack of formal validation of domain mapping limit generalizability and measurement power. Therefore, future studies should develop a validated Perspective Switching instrument, systematically record the roles and experiences of respondents, and use experimental designs to test whether perspective switching instructions or training can improve software issue detection. Further studies could also compare conventional assessments with assessments that explicitly guide the switching between perspectives, particularly when evaluating software that is generated or assisted by AI. Such an approach could determine whether perspective diversity is not only inherent in the consideration process but contributes to a more comprehensive software evaluation.

Declaration of Ethics

Participation in this study was voluntary, and respondents were explained the purpose of the study before answering the questionnaire. No personally identifiable data (such as name or contact information) was collected; all responses were recorded anonymously using respondent codes (e.g. R01, R02).

Data Availability

Anonymized quantitative data sets supporting the findings of this study can be provided by the corresponding author upon reasonable request.

Conflict of Interest

The authors declare no conflicts of interest related to the publication of this study.

REFERENCES

1. Brasoveanu, A., Moodie, M., & Agrawal, R. (2020). Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study. *CEUR Workshop Proceedings*, 2657, 1–9. <https://doi.org/10.1145/nnnnnnnn.nnnnnnn>
2. Gupta, S., Epiphaniou, G., & Maple, C. (2023). AI-augmented usability evaluation framework for software requirements specification in cyber physical human systems. *Internet of Things (Netherlands)*, 23. <https://doi.org/10.1016/j.iot.2023.100841>
3. Gurunath Sakhalkar, P. (2025). AI-Assisted Code and Vulnerability Management: Transforming Modern Software Development. <https://doi.org/10.32996/jcsts>
4. Hu, X., Chen, Q., Wang, H., Xia, X., Lo, D., & Zimmermann, T. (2022). Correlating Automated and Human Evaluation of Code Documentation Generation Quality. *ACM Transactions on Software Engineering and Methodology*, 31(4). <https://doi.org/10.1145/3502853>
5. Kauppinen, T. (2026). Generative Artificial Intelligence and Expert Work: Designing a Dynamic AI Capability.
6. Majdinasab, V., Bishop, M. J., Rasheed, S., Moradidakhel, A., Tahir, A., & Khomh, F. (2023). Assessing the Security of GitHub Copilot Generated Code -- A Targeted Replication Study. <http://arxiv.org/abs/2311.11177>

7. Melo, A., Fagundes, R., Lenarduzzi, V., & Santos, W. B. (2022). Identification and measurement of Requirements Technical Debt in software development: A systematic literature review. *Journal of Systems and Software*, 194. <https://doi.org/10.1016/j.jss.2022.111483>
8. Morgan, S., & Jamison, J. (2025). The Effect of Ai Code Generation on Developer Productivity I The Effect of Artificial Intelligence Code Generation on Software Developer Productivity.
9. Oruthotaarachchi, C. R., & Wijayanayake, W. M. J. I. (2025). Systematic Review of Software Product Value: Perspectives Beyond Functionality. In *Information and Software Technology* (Vol. 185). Elsevier B.V. <https://doi.org/10.1016/j.infsof.2025.107784>
10. Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do Users Write More Insecure Code with AI Assistants? <https://anonymous.4open>.
11. Roberts, C., & Badi, S. (2026). Artificial Intelligence and Secure Coding Practices: A Comparative Analysis of AI-Assisted Development Platforms.
12. Rojas, A. M., Amaya, S. P., & Calderon, J. M. (2022). PBR Compiler: Automatic system description generator software. *IFAC-PapersOnLine*, 55(4), 219–224. <https://doi.org/10.1016/j.ifacol.2022.06.036>
13. Shethiya, A. S. (2025). AI-Assisted Code Generation and Optimization in .NET Web Development. In *Annals of Applied Sciences Annal. App. Sci* (Vol. 6, Number 1). <https://annalsofappliedsciences.com>
14. Tuomimaa, J. (2025). Generative AI as a Source of Competitive Advantage in Decision-Making Processes: Dynamic Capability View.
15. Yilmaz, R., & Karaoglan Yilmaz, F. G. (2023). Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning. *Computers in Human Behavior: Artificial Humans*, 1(2), 100005. <https://doi.org/10.1016/j.chbah.2023.100005>
16. Yoo, S. H., & Kim, H. J. (2025). Security Analysis of Automated Code Generation: Structural Vulnerabilities in AI-Generated Code. *Tehnicki Glasnik*, 19(4), 560–574. <https://doi.org/10.31803/tg-20250225095135>