

Human-AI Work Redistribution Across the Software Development Life Cycle: A Socio-Technical Study of Indonesian Developers

Arisantoso Arisantoso^{1*}, Achmad Benny Mutiara Qn², Abdullah Said Alkalbani³, Saipul Anwar⁴,
Muhamad Femy Mulya⁵

Abstract—AI Coding Assistants are becoming increasingly integrated into software development, changing not only how developers perform their tasks but also how humans and AI share responsibilities. However, how this redistribution of work occurs across the Software Development Life Cycle (SDLC) remains insufficiently understood. This study investigates how responsibilities are distributed between developers and AI across SDLC activities, how these changes affect software engineering workflows and developer roles, and which socio-technical factors influence this process. Semi-structured interviews with seven software professionals in Indonesia were analyzed using reflexive thematic analysis. The results identify three patterns of human-AI work redistribution: AI-delegated, AI-augmented, and human-controlled work. AI contributes more substantially to structured, repetitive, and relatively verifiable activities, particularly code generation, debugging, testing, and documentation. In contrast, requirements analysis, architectural decisions, security assessment, and release approval remain largely under human control because they require contextual understanding, professional judgment, and accountability. The findings also indicate a shift in developer responsibilities. This redistribution depends on developer experience, AI literacy, trust, model capability, task complexity, security requirements, team practices, and organizational policies. Drawing on socio-technical systems theory and activity theory, the study develops an empirically grounded Human-AI Work Redistribution Model that conceptualizes AI-assisted software engineering as a selective

redistribution of responsibilities rather than uniform automation across the SDLC.

Index Terms—AI coding assistant, human-AI collaboration, human-AI work redistribution, SDLC, socio-technical systems.

I. INTRODUCTION

The rapid progress of Artificial Intelligence (AI), especially Large Language Models (LLMs), has changed the way software engineering is practiced. Today's AI Coding Assistants go beyond the traditional programming support tools that focus on syntax completion and code search, and can generate source code, explain programming logic, help with debugging, generate test scenarios, and support technical documentation. These developments have made AI more than a simple programming assistant; it is an interactive partner that participates in many activities in software engineering [1]–[3].

Recent studies show that AI Coding Assistants offer substantial opportunities to improve the efficiency of software development. Tools built on LLMs can reduce repetitive programming work, accelerate the exploration of technical solutions, and help developers solve programming problems more efficiently [4], [5]. Research on the application of LLMs to software engineering also indicates that standard artificial intelligence can assist with a variety of activities, such as code generation, debugging, testing, and documentation. These benefits, however, rely heavily on prompt quality, developer experience, and developers' ability to evaluate AI-generated outputs [2], [6].

These advantages notwithstanding, AI adoption in software engineering cannot be seen through the lens of productivity. Software development is a complex activity that encompasses technical processes, human expertise, organizational rules, and quality assurance mechanisms. Previous work has shown that AI-generated code can contain security vulnerabilities, incorrect assumptions, inappropriate dependencies, and solutions that are not fully aligned with system requirements [7], [8]. AI adoption is thus an ongoing human activity that involves

Received: 6 August 2026; Revised: 5 September 2026; Accepted: 3 October 2026.

*Corresponding author

¹Arisantoso Arisantoso, Dept. of Information Systems, Faculty of Engineering and Technology, Tanri Abeng University, Indonesia (e-mail: arisantoso@tau.ac.id).

²Achmad Benny Mutiara Qn, Postgraduate in Information Technology, Faculty of Computer Science and Information Technology, Gunadarma University. (e-mail: amutiara@staff.gunadarma.ac.id).

³Abdullah Said Alkalbani, College of Engineering, University of Buraimi Oman (e-mail: abdullah.s@uob.edu.om).

⁴Saipul Anwar, Dept. of Information Systems, Faculty of Engineering and Technology, Tanri Abeng University, Indonesia (e-mail: saipul@tau.ac.id).

⁵Muhamad Femy Mulya, Dept. of Information Systems, Faculty of Engineering and Technology, Tanri Abeng University, Indonesia (e-mail: femy.mulya@tau.ac.id).

checking generated outputs, validating software logic, ensuring security, and making architectural decisions.

Previous research on AI-assisted software engineering has examined various aspects of AI use, including code generation, usability, developer productivity, security, and specific development activities. However, we still have limited understanding of how the increasing use of AI Coding Assistants changes the distribution of responsibilities between developers and AI across different SDLC activities. It also remains unclear how these changes affect software engineering workflows and reshape developers' roles. Furthermore, existing studies have paid relatively little attention to the socio-technical conditions that shape when, where, and to what extent developers and AI share or redistribute responsibilities [7], [8], [6]. Similarly, research on developer perceptions and factors influencing AI adoption has mainly examined acceptance, usability, and organizational readiness, but not changes in software engineering practices across SDLC stages [9], [10].

Moreover, the existing evidence on AI-aided software engineering comes mainly from industrial software engineering environments in developed countries. However, the adoption of AI will vary depending on the organization's readiness, technology infrastructure, security governance, and developer competencies. In Indonesia, software developers are found across different organizational contexts, with varying levels of technical capacity, workplace policies, and professional practices that influence the acceptance of AI [9], [10]. Therefore, understanding the transformation of AI-assisted workflows in the Indonesian context makes an important contribution to extending current software engineering research.

To fill this gap, this study adopts a socio-technical perspective. Socio-technical systems theory suggests that technological transformation results from the interaction between technical components and social components, such as people, organizational structures, processes, and work practices. In this sense, AI Coding Assistants should not be regarded as mere productivity tools but as technological artifacts that alter how software developers perform tasks, collaborate, and make decisions.

In addition, the study uses Activity Theory as an analytical framework to examine the changes that AI brings to the software engineering activity system. Activity Theory depicts work as a set of interrelated components, including the subject, object, mediating artifacts, rules, community, and division of labor. In AI-assisted Software Engineering, the developer is the subject, the software delivery is the object, and AI Coding Assistants are mediating artifacts. AI adoption also changes the rules of work, how we work together, and how we share responsibility between humans and AI. This results in a core contradiction: while artificial intelligence can accelerate software development activities, increased automation also demands improved validation practices, critical judgment, and the retention of developer expertise.

Recent work on human-AI collaboration shows that AI does not replace software developers but changes their role from

traditional code producers to AI coordinators, reviewers, and decision-makers [11], [12]. This shift implies that the future of software engineering will increasingly rely on developers' ability to work alongside AI while maintaining accountability for software quality, security, and ethical concerns.

This study aims to address this theoretical-empirical gap by investigating the impact of AI Coding Assistants on the software engineering workflow of software developers in Indonesia. In particular, this research addresses three research questions.

- RQ1: How are responsibilities redistributed between developers and AI Coding Assistants across different stages of the SDLC?
- RQ2: How does this human-AI redistribution of work transform software engineering workflows and developer roles?
- RQ3: What socio-technical factors enable or constrain the redistribution of human-AI work in software engineering?

The present study contributes to the extant literature in three ways. First, it extends prior work on AI-assisted software engineering by examining how the workflow transforms across multiple SDLC stages, rather than focusing solely on coding activities. Second, it uses socio-technical systems theory and Activity Theory to explain how AI adoption changes the relationships among developers, technological artifacts, organizational rules, and the division of labor. Third, it proposes an empirically grounded Human-AI Work Redistribution Model that explains how responsibilities shift between developers and AI across software engineering activities and how socio-technical conditions shape this redistribution.

II. RELATED WORK

A. AI Coding Assistants in Software Engineering Transformation

Large Language Models (LLMs) have significantly changed software engineering practices with the introduction of AI Coding Assistants, which can assist with various development activities, including code generation, explanation, debugging, testing, and documentation. Unlike traditional programming support tools that mainly provide syntax completion and code retrieval, LLM-based assistants enable interactive collaboration between developers and artificial intelligence systems via natural language interaction [13], [14].

Recent empirical studies indicate that AI Coding Assistants can enhance developer productivity by alleviating repetitive programming effort and quickening solution exploration. Reference [5] examined the usability of AI-based code-generation tools. They found that developers gain from faster solution discovery but still struggle with the comprehension, verification, and integration of generated code into existing systems. Likewise, Barke et al. showed that developers engage with code-generating models in iterative cycles of exploration, prompting, evaluation, and refinement rather than simply substituting code [14].

However, the code generated by the AI raises a range of

technical and managerial issues. Studies show that generated code can contain security vulnerabilities, incorrect assumptions, and implementation decisions that do not align with software requirements [15], [16]. Prior study showed that AI programming assistants can produce insecure code patterns, highlighting the need for developer review and security awareness when using AI-based development tools [15].

Moreover, research on LLM applications in software engineering indicates that current AI capabilities remain focused on specific development tasks, such as code generation, debugging, and testing assistance. In contrast, the broader transformation of Software Development Life Cycle (SDLC) workflows has been less studied [17], [18]. While previous work demonstrates the technical feasibility of AI Coding Assistants, additional research is required to understand how AI alters software engineering activities, developer responsibilities, and workflow structures.

B. AI-Assisted Software Development Across the Software Development Life Cycle (SDLC)

The adoption of AI is not uniform across the SDLC activities in software engineering. Current studies show that AI is better at assisting with activities that involve structured patterns and measurable outputs, while activities requiring contextual interpretation and strategic judgment still heavily depend on human expertise [17], [19].

1) Requirements engineering

Requirements engineering requires communication with interested parties, business understanding, and the interpretation of complex user needs. Although LLMs can help developers write user stories, summarize requirements, and create initial specifications, current studies suggest that AI is still limited in handling ambiguity, negotiation, and organizational context [18]. This limitation is because requirements engineering is not only an information-processing activity but also a socio-organizational process involving human interaction and decision negotiation. Therefore, AI's current role is to serve as an augmentation mechanism rather than a replacement for human analysts [19].

2) Software design and architecture

The application of AI in software design is attracting growing research attention due to the potential of LLMs to generate architectural alternatives, explain design patterns, and support technical exploration. However, architectural decisions must take into account scalability, security, maintainability, cost, and organizational constraints that cannot be fully captured by AI-generated recommendations [17], [20]. A previous study emphasized that organizations need to establish proper validation processes, since AI outputs may not reflect the context of system-level decisions, despite LLMs being beneficial in supporting software engineering activities [18].

3) Implementation, testing, and maintenance

Implementation is the software engineering activity that has seen the most AI adoption, as source code generation maps directly to what LLMs can do. According to research, AI assistants can reduce coding effort, help with debugging, and improve developer productivity [13], [16]. AI-assisted testing is another important research area, since LLMs can generate test cases, explain failures, and support debugging. However, developers are responsible for verifying that the generated tests sufficiently reflect functional requirements and possible edge cases [20].

Artificial intelligence can help in code summarization, documentation generation, and legacy system analysis for maintenance activities. However, human expertise remains essential because maintenance decisions are often informed by historical knowledge and organizational context [17].

These results imply that AI involvement is not uniform across SDLC stages, which calls for workflow-oriented analysis rather than task-oriented evaluation.

C. Human-AI Collaboration and Transformation of Developer Roles

The advent of AI Coding Assistants has revolutionized the traditional role of software developers. In the past, software code was largely written by hand by developers. As AI adoption increases, developers are increasingly engaged in prompting, evaluating AI outputs, integrating generated solutions, and making engineering decisions [14], [19].

Reference [19] argued that human-AI collaboration in software engineering is a progression from automation toward augmentation, with AI augmenting human capability. In contrast, humans are responsible for judgment and accountability. This transformation requires new competency needs for software developers. More and more developers need to have skills in:

- prompt engineering;
- AI output evaluation;
- critical reasoning;
- security assessment;
- AI governance.

Research indicates that the success of collaboration with AI depends on developers' ability to understand AI's boundaries and calibrate trust in its recommendations [21]. However, greater reliance on artificial intelligence may also create risks. Automation bias may lead users to accept AI-generated recommendations without proper verification, thereby affecting software reliability and security [22]. Therefore, AI adoption implies a balanced collaboration in which AI enhances human capabilities rather than substituting for professional judgment.

D. Trust, Security, and Governance in AI-Assisted Software Engineering

Trust is the foundation of successful AI adoption. Developers need to build the right level of trust in artificial

intelligence (AI) systems, as both under- and over-trust can negatively affect software development outcomes [21].

Studies in automation have shown that users struggle to determine when to trust or distrust an automated system. A prior study found that automated systems may be subject to misuse, disuse, or abuse, resulting in inappropriate reliance on automation [22].

Trust calibration is particularly important in AI-assisted software engineering, where AI-generated outputs can be convincing but contain hidden errors. Hence, developers need mechanisms to assess AI recommendations before integrating them into production systems [15], [23].

In addition to individual trust, organizations need to establish governance frameworks that address the protection of confidential information, intellectual property rights, software licensing, security validation, and accountability.

Other studies have successfully adopted LLMs in software engineering, requiring organizational strategies that balance productivity improvements with software quality and engineering responsibility [17].

E. Socio-Technical Systems Perspective in AI-Assisted Software Engineering

Socio-technical systems theory provides a theoretical basis for understanding AI adoption as an interaction of technical and social elements. Unlike technology-centered perspectives, socio-technical perspectives argue that successful innovation depends on the alignment of technology, people, organizational structures, and work practices [24]. Another concept holds that socio-technical systems comprise technical artifacts and human and organizational elements, as system performance arises from their interaction rather than from technology alone [24]. In AI-assisted software engineering, the technical subsystem includes AI Coding Assistants, development environments, software platforms, and computational infrastructure. Meanwhile, the social subsystem includes developers, engineering teams, organizational policies, and professional practices.

This explains why the same artificial intelligence (AI) tools can produce different results across organizations. The effectiveness of AI integration depends on differences in developer capability, organizational readiness, governance maturity, and engineering culture [25]. Thus, this study adopts a socio-technical perspective to analyze AI Coding Assistants as change agents within software engineering work systems, rather than as stand-alone technological tools.

F. Activity Theory Perspective on Workflow Transformation

Activity theory is a complementary framework for studying the transformation of human activities through technological innovation. Engeström's concept of activity systems is based on interconnected elements: the subject, the object, mediating artifacts, rules, the community, and the division of labor [26]. In AI-assisted software engineering, these components can be modeled as shown in Table 1.

Table 1.

AI-Assisted Software Engineering

Activity Component	AI-Assisted Software Engineering Context
Subject	Software developers
Object	Software development and delivery
Mediating artefact	AI Coding Assistants
Rules	Coding standards, security policies, organizational guidelines
Community	Software teams and stakeholders
Division of labor	Distribution of responsibilities between humans and AI

Kaptelinin and Nardi described how technological artifacts change human activities by altering interaction patterns, responsibilities, and problem-solving [27]. The introduction of AI into the activity system of AI Coding Assistants creates a contradiction within it. On one hand, AI brings efficiency by eliminating repetitive programming tasks. On the other hand, developers need to put more effort into verification, monitoring, and keeping their expertise up to date. This contradiction points to a fundamental difficulty in AI-assisted software engineering: how to gain productivity without sacrificing human accountability and professional competence.

G. Research Gap and Positioning of Current Study

From previous studies, there are some research gaps. Existing works mainly focus on task-level evaluation of AI Coding Assistants, especially for code generation, debugging, and productivity measurement [13], [23], [18]. Previous research offers limited empirical insight into how developers and AI share responsibilities across SDLC activities, how this shifts developer roles, and how socio-technical conditions shape the distribution of work. Secondly, prior research often concentrates on technical performance. However, it offers little explanation of shifts in developer roles, organizational practices, and human-AI collaboration [19], [25]. Third, most empirical studies are conducted in technologically mature software ecosystems. Evidence from emerging contexts, including Indonesia, remains limited despite differences in organizational readiness, governance practices, and developer competences. Fourth, current conceptual workflow models are often not empirically linked to developer experiences.

To address these limitations, this research presents an AI-assisted software engineering workflow model based on qualitative evidence from Indonesian software developers. This work integrates socio-technical systems theory and activity theory to characterize how AI Coding Assistants affect SDLC activities, reallocate developer responsibilities, and reframe human-AI collaboration (Table 2).

Table 2.

Comparative Literature Synthesis

Ref.	Year	Venue	Method	Focus	Scope Relative to Current Study
[5]	2022	CHI EA	User Study	Usability of LLM code generation tools	Focused on code-generation interaction
[4]	2023	PACMPL	Grounded	Programm	Focused on

Ref.	Year	Venue	Method	Focus	Scope Relative to Current Study
		OOPSLA 1	theory/ observational study	er interaction with LLM	programming tasks rather than the broader SDLC Security-focus d tasks
[2]	2023	ACM CCS	User study/ security experiment	Security of AI-assisted code	
[28]	2023	arXiv	Controlled experiment	Developer productivity with GitHub Copilot	Controlled programming task
[7]	2023	IEEE Software	Perspective/ conceptual article	Opportunities, risks, and implications of LLMs for SE	Conceptual rather than empirical
[29]	2023	IEEE Software	Conceptual/ vision	Human-centered AI-assisted software management	Requires empirical validation
[30]	2024	ACM TOSEM	Systematic Literature Review	LLM applications across software engineering	Broad synthesis rather than end-to-end workflow evidence
[31]	2026	SCAM 2026 / arXiv	Empirical study	LLM-based test generation	Focused specifically on software testing
Current study		AISM	Qualitative interview study	Human-AI work redistribution across the SDLC	Context-specific sample; broader validation needed

III. RESEARCH METHOD

A. Research Design

This study used an exploratory qualitative interview approach, supported by reflexive thematic analysis, to explore how AI Coding Assistants are transforming software engineering practices and workflows in Indonesia. Rather than conceptualize developers as independent analytical units, this study positions their lived experiences as a critical source of contextual knowledge for understanding AI adoption processes, changes in software development practices, and the reconfiguration of human–AI collaboration. This approach was considered appropriate because AI integration in software engineering is not merely a technical transition but a multifaceted process that involves the interplay among technological artifacts, organizational environments, developer capabilities, and professional judgment. Therefore, productivity metrics alone cannot fully capture its implications, as they encompass broader changes in work practices, roles, and decision-making processes.

This research drew on two complementary theoretical perspectives: Socio-technical systems theory and activity theory. From a socio-technical standpoint, AI Coding Assistants were understood as technological artifacts embedded within a complex work system comprising human actors, organizational norms, software engineering practices, and governance structures. This perspective enables examination of AI adoption as an interaction between technological capabilities and the social environment in which development activities take place. Building upon this foundation, activity theory was employed as an analytical lens to investigate transformations within the software engineering activity system. Specifically, the framework examined how AI Coding Assistants influence developers' practices, the distribution of tasks and responsibilities, collaboration patterns, and the evolving relationship between human expertise and AI-supported activities (Table 3).

Table 3.
Application of Activity Theory to Examine AI-Assisted Transformations in Software Engineering Workflows

Activity Element	AI-Assisted Software Engineering Context
Subject	Software developers
Object	Software development and delivery outcomes
Mediating artefact	AI Coding Assistants (GitHub Copilot, ChatGPT, Gemini, CodeWhisperer, Tabnine)
Rules	Coding standards, security policies, organizational guidelines
Community	Development teams, managers, stakeholders
Division of labor	Redistribution of responsibilities between humans and AI

The key contradiction revealed through the analysis reflects the transformative yet paradoxical nature of AI integration in software engineering. Although AI Coding Assistants boost productivity by accelerating development and reducing repetitive work, their adoption also reinforces the importance of human cognitive capabilities, including critical judgment, contextual validation, and the preservation of essential engineering expertise.

B. Participants and Sampling

Participants were selected using purposive sampling to include individuals with rich, relevant experience in AI-supported software development. The determination of sample adequacy was informed by the concept of information power, which emphasizes that the size of a qualitative sample should be evaluated by the strength and depth of information participants can contribute rather than by numerical representativeness. In this study, information power was assessed through the specificity of participants' experiences, the study's focused research objectives, theoretical assumptions, the quality of empirical engagement, and the analytical framework employed.

The study involved seven software professionals recruited according to predefined eligibility criteria to ensure relevant,

experience-rich insights into AI-assisted software development. The inclusion criteria required participants to:

- Hold professional roles related to software development, including software developers, software engineers, technical architects, programmers, or DevOps engineers;
- Possess a minimum of two years of industry experience in software development;
- Demonstrate sustained use of AI Coding Assistants for at least six months; and
- Have practical experience integrating AI tools throughout different stages of the Software Development Life Cycle (SDLC).

The participants came from a range of technology-driven sectors, including fintech, e-commerce, logistics, digital healthcare, financial services, cloud platforms, and media. Given the sensitivity of professional experiences and organizational contexts, the authors protected participants' identities using anonymized codes (I1–I7).

C. Data Collection

The authors collected empirical data through semi-structured online interviews conducted between June and July 2026. Each interview lasted 60 to 90 minutes, providing sufficient depth to capture participants' experiences, interpretations, and reflections on the adoption of the AI Coding Assistant. The interview questions were structured around seven thematic domains: initial encounters with AI Coding Assistants, AI integration across SDLC activities, workflow transformation, perceived effects on productivity and quality, human oversight mechanisms, security and governance challenges, and the future evolution of software engineering practices.

Interviews were conducted in Bahasa Indonesia, the participants' primary language, to facilitate richer narratives and minimize potential communication barriers. After transcription, the interview data were translated into English for reporting. Translation reliability was supported through iterative researcher checking, comparing the original transcripts with the English translations to ensure conceptual equivalence and contextual accuracy.

D. Data Analysis and Trustworthiness

The interview data were analyzed using reflexive thematic analysis, following the six-phase framework proposed by Braun and Clarke. This approach enabled an in-depth interpretation of participants' experiences and identified meaningful patterns related to AI adoption and workflow transformation in software engineering contexts.

The analytical process involved six iterative stages:

- Familiarization with the interview transcripts through repeated reading;
- Generation of initial codes representing relevant patterns within the data;
- Construction of preliminary themes from related codes;
- Critical review and refinement of emerging themes;
- Defining and naming themes to capture their conceptual meanings; and

- Developing the final analytical narrative that connected empirical findings with the research objectives and theoretical perspectives.

NVivo 12 supported data management, transcript organization, and systematic coding throughout the analytical process. The coding strategy was both theoretically informed and empirically driven, with interpretations guided by the research questions and the selected theoretical frameworks. Specifically, socio-technical systems theory informed the analysis of relationships among technological artifacts, human actors, and organizational structures. At the same time, Activity Theory supported the examination of transformations in tools, rules, and the division of labor within software engineering activities. Participant feedback was sought on selected contextual interpretations to clarify meaning and reduce the risk of misrepresenting participants' accounts.

This study does not seek statistical generalization. Instead, it aims to provide analytical generalization by offering theoretically informed insights applicable to similar contexts involving AI-assisted software development. The complete research methodology and analysis process are shown in Fig. 1.

IV. RESULT AND DISCUSSION

A. Overview of Emerging Themes

The reflexive thematic analysis produced four analytical themes that illustrate how software engineering activities have transformed following the adoption of AI Coding Assistants. Collectively, these themes highlight changes in development practices, human–technology relationships, professional roles, and the broader socio-technical dynamics surrounding AI integration in software engineering environments (Table 4).

Table 4.
Emerging Themes and Analytical Insights from Reflexive Thematic Analysis of AI-Assisted Software Development

Theme	Definition	Supporting Informants
AI as an acceleration mechanism	AI reduces repetitive effort and expands solution exploration	I1, I3, I4, I5
Human validation as a mandatory activity	AI outputs require review, testing, and contextual judgment	I2, I3, I5, I7
Governance and trust calibration	AI adoption depends on security, privacy, and organizational rules	I5, I6, I7
Developer role transformation	Developers shift from code producers toward AI coordinators and reviewers	I1–I7

B. Human-AI Work Redistribution Across SDLC Activities

The analysis showed that AI involvement varies across SDLC activities. Participants commonly used AI Coding Assistants to reduce repetitive work, generate initial outputs, and explore possible solutions. As I1 explained, "It is like having a personal AI assistant that is always ready to type source code." However, AI was more involved in structured, repetitive tasks, while activities requiring contextual understanding, technical judgment, and accountability remained largely dependent on developers. Table 5 shows how

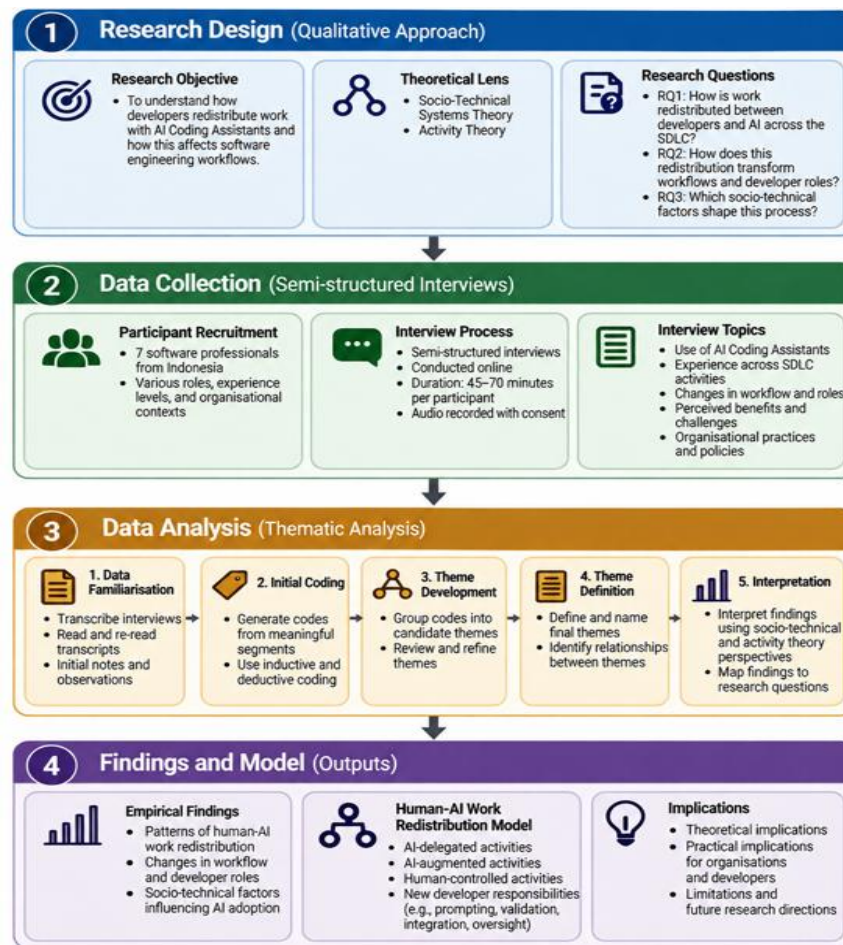


Fig. 1. Research methodology and analysis process.

this distribution varies across the SDLC. AI plays a greater role in tasks such as boilerplate code generation, initial testing, and documentation. In contrast, requirements analysis, architectural decisions, security assessment, and release approval remain largely human-controlled because they require contextual judgment, risk assessment, and professional accountability.

C. Developer Role and Workflow Transformation

The findings show that the redistribution of human-AI work also reshapes developers' roles in software engineering. As AI takes a greater role in generating initial code, tests, documentation, and potential solutions, developers shift some effort from direct production to providing context, guiding AI interactions, reviewing outputs, and integrating results into existing systems. Human validation remains essential throughout this process. Participants stressed that teams must review AI-generated outputs before using them in a project. Developers remain responsible for ensuring that these outputs are technically appropriate, secure, and aligned with project requirements. This responsibility matters most in architecture, security, and production deployment, where errors can have wider consequences. Overall, developers increasingly act as coordinators and evaluators of AI-supported work. Prompt

formulation, contextualization, output verification, refinement, integration, and final decision-making are becoming key parts of their responsibilities. AI adoption, therefore, changes both how development tasks are performed and where developers focus their efforts within the workflow.

Table 5 shows that the distribution of human-AI work varies across SDLC activities. AI plays a larger role in structured, repetitive tasks such as code generation, initial testing, and documentation. In debugging, code refinement, testing, and maintenance, developers and AI work more collaboratively, with developers reviewing and refining AI outputs. Requirements analysis, architectural decisions, security assessment, and release approval remain largely human-controlled because they require contextual judgment, risk assessment, and accountability. Overall, AI involvement varies with task characteristics and the level of human judgment required.

Figure 2 summarizes three patterns of human-AI work redistribution: human-controlled, AI-augmented, and AI-delegated work. These patterns reflect varying levels of AI involvement, from human-led decisions to collaborative tasks and AI-led initial execution, while developers retain responsibility for outcomes.

Table 5.
Human-AI Work Redistribution Across SDLC Activities

SDLC Activity	AI Contribution	Human Responsibility	Redistribution Pattern
Requirements analysis	Supports requirement clarification, summarization, and exploration of alternatives	Interprets business context, stakeholder needs, constraints, and final requirements	Human-controlled
System design and architecture	Suggests design alternatives, patterns, and possible technical solutions	Evaluates trade-offs, system constraints, scalability, and architectural decisions	Human-controlled/ AI-augmented
Coding and implementation	Generates boilerplate code, code suggestions, and initial implementations	Provides context, reviews generated code, adapts outputs, and approves integration	AI-augmented / partially AI-delegated
Debugging and code refinement	Suggests possible causes, fixes, refactoring options, and code improvements	Diagnoses contextual problems, verifies proposed fixes, and evaluates side effects.	AI-augmented
Testing	Generates initial test cases and supports test improvement	Defines testing priorities, validates coverage, interprets failures, and verifies quality	AI-augmented/ partially AI-delegated
Documentation	Produces initial technical documentation, summaries, and code explanations	Reviews accuracy, adds project context, and approves final documentation	Partially AI-delegated
Security assessment	Identifies potential issues and suggests checks or remediation options	Evaluates security implications, verifies risks, and remains accountable for decisions	Human-controlled/ AI-augmented
Deployment and release	Supports configuration guidance, troubleshooting, and deployment preparation	Controls release decisions, evaluates operational risk, and approves deployment.	Human-controlled
Maintenance and monitoring	Supports log interpretation, issue diagnosis, and possible corrective actions	Evaluates production context, validates changes, and remains responsible for system operation	AI-augmented

D. Socio-Technical Conditions Shaping Work Redistribution

The interviews showed that AI involvement depends on more than the technical capabilities of AI Coding Assistants.

Developer experience and AI literacy influenced how participants formulated prompts, evaluated outputs, and identified potential errors. Trust also depended on the reliability of the output, particularly on whether participants could review and verify AI-generated results. Task characteristics further shaped AI involvement. Structured, well-defined tasks were better suited to AI support, while activities involving ambiguity, domain knowledge, security concerns, or complex trade-offs required greater human involvement. Team practices, privacy and security requirements, and organizational policies also influenced how teams used AI tools. Overall, the redistribution of human-AI work is shaped by human, technological, task-related, and organizational conditions. As a result, the level of AI involvement may vary across developers, projects, and organizations, even when using similar AI tools.

E. Development of The Human-AI Work Redistribution Model

The proposed model emerged from themes identified through reflexive thematic analysis and was interpreted using socio-technical systems theory and activity theory. Activity theory helps explain how AI reshapes the division of labor and developer responsibilities. In contrast, socio-technical systems theory explains how human, technological, task-related, and organizational conditions shape this redistribution. The model integrates three main findings: variations in AI involvement across SDLC activities, shifts in developer responsibilities, and the conditions that influence the redistribution of human-AI work. Table 6 shows how these elements emerged from participants' accounts. The model represents an empirically grounded conceptualization of participants' experiences, not a universally validated model of AI-assisted software development.

Table 6.
Empirical Reconstruction of the AI-Assisted Software Development Model from Emerging Themes

Model Element	Empirical Basis
Prompt Engineering Stage	Developers reported needing improved interaction strategies with AI
AI Code Generation	Frequent AI use in coding activities
Human Review Gate	Participants emphasized verification necessity
Code Refinement	Developers modify AI outputs before integration
AI-Assisted Testing	AI supports test creation but requires human evaluation
Monitoring and Maintenance	Human accountability continues after deployment

Figure 3 integrates the findings into a conceptual model of human-AI work redistribution in software engineering. It links the three redistribution patterns to changes in developer responsibilities and to the conditions shaping AI involvement. As AI becomes more involved in development tasks, developers increasingly focus on contextualization, validation, integration, and oversight, while retaining responsibility for activities involving judgment, risk, security, and accountability.

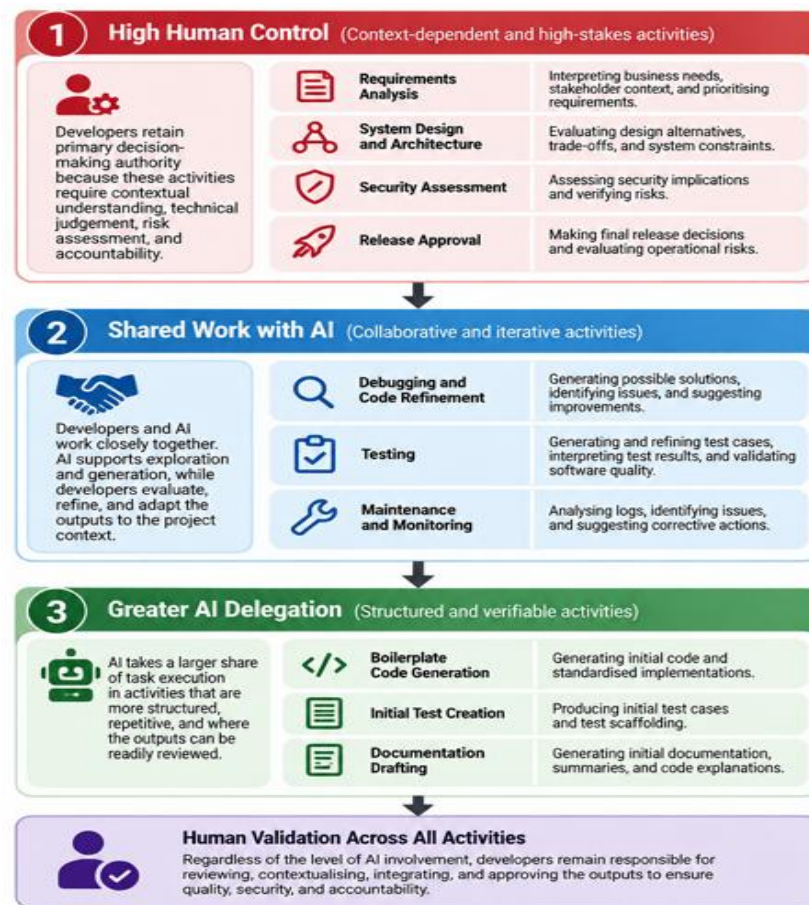


Fig. 2. Patterns of human-AI work redistribution across SDLC activities.

F. Discussion

The adoption of AI Coding Assistants appears to affect not only the speed of software development but also how responsibilities are shared between developers and AI throughout the Software Development Life Cycle (SDLC). This change varies across development activities. Instead, AI involvement depends largely on the task's nature, complexity, and the level of human judgment required. Previous studies have mainly examined AI assistance within particular development tasks, such as programmers' interactions with code-generating models [5], developer productivity [2], and AI-supported software testing [7]. While these studies show AI's usefulness in specific activities, this study offers a broader view by showing how AI assistance contributes to the redistribution of work across different stages of the SDLC.

The results reveal a clear pattern in how responsibilities are distributed between developers and AI. AI tends to play a larger role in structured, repetitive, and relatively easy-to-verify activities, particularly code generation, debugging, testing, and documentation. This pattern aligns with previous research showing that AI Coding Assistants can support programming activities and reduce the effort required for certain development tasks [5], [2]. Study [31] also demonstrates the potential of LLM-based assistance in software testing. However, AI

involvement becomes more limited when a task requires deeper contextual understanding, consideration of technical trade-offs, security judgment, or professional accountability. Activities such as requirements analysis, architectural decision-making, security assessment, and release approval therefore continue to rely heavily on human expertise. These results indicate that AI adoption across the SDLC does not lead to uniform automation. Instead, it selectively redistributes work based on the characteristics and demands of each activity.

This redistribution also changes what developers do within the development process. When AI contributes to code generation, testing, documentation, and producing candidate solutions, developers increasingly focus on providing relevant context, directing interactions with AI, evaluating generated outputs, integrating appropriate results, and making final engineering decisions. This pattern corresponds with the human-centered perspective on AI-assisted software management proposed by [29], which emphasizes collaboration between human expertise and AI capabilities. This study extends that perspective by showing how such collaboration takes shape across different software engineering activities. The use of AI does not simply reduce developers' workload. Instead, it shifts part of their effort from direct

production toward coordination, validation, integration, and oversight.

Within this changing workflow, human validation becomes increasingly important. AI can accelerate the production of an initial solution. However, developers still need to assess whether the generated output is technically correct, secure, relevant to the project context, and appropriate for integration into the existing system. Reference [7] similarly highlights the opportunities and risks of using large language models in software engineering. These concerns are also reflected in the broader review by [30], which shows that LLM applications have expanded across numerous software engineering activities while continuing to raise questions about reliability, quality, and responsible use. This study's results indicate that AI may reduce effort for production-oriented tasks while making verification, refinement, and integration more important. Therefore, improvements in the speed or efficiency of individual tasks should not automatically be interpreted as equivalent improvements in the overall development workflow.

Activity theory provides a useful perspective for explaining this change. Within the activity system adopted in this study, developers remain the subjects of software development activities. At the same time, AI Coding Assistants act as mediating artifacts that influence those activities carried out. Their introduction changes the division of labor because AI can partially delegate or augment some activities. At the same time, developers continue to control decisions that involve substantial judgment, risk, and accountability. This interpretation is consistent with Activity Theory, which explains how technological artifacts can reshape patterns of interaction, responsibility, and problem-solving within an activity system [25], [32]. From this perspective, the importance of AI adoption lies not only in its ability to generate integration, and oversight. Workflow transformation therefore involves not only redistributing existing tasks but also creating new responsibilities for managing and evaluating AI

code or other outputs but also in how its use organizes human responsibilities within software engineering practice.

A socio-technical perspective further explains why AI involvement varies across development activities and organizational contexts. The results indicate that several interconnected factors influence the redistribution of work, including developer experience, AI literacy, trust, model capability, output reliability, task complexity, team practices, security requirements, and organizational policies. This observation is consistent with the socio-technical view that the performance and use of a technological system cannot be understood separately from the people, work practices, and organizational environment in which the technology operates [23]. As a result, individual developers, teams, and organizations may adopt and use the same AI Coding Assistant in different ways. The distribution of work between humans and AI therefore depends not only on what the technology can do, but also on the social, technical, and organizational conditions that shape its use.

Taken together, these results provide empirical support for the Human-AI Work Redistribution Model presented in Figure 3. The model distinguishes three broad patterns of work: AI-delegated, AI-augmented, and human-controlled activities. In AI-delegated activities, AI takes a larger role in generating an initial output that developers subsequently review or refine. AI-augmented activities involve more iterative collaboration, where developers and AI contribute throughout the task. Human-controlled activities remain primarily the developer's responsibility because they involve contextual judgment, risk assessment, or accountability that cannot be readily transferred to AI. Importantly, the model also identifies new forms of human work that emerge from AI adoption, including prompting, contextualization, output verification, contributions.

The Indonesian context adds another dimension to these findings. Previous studies have produced substantial evidence

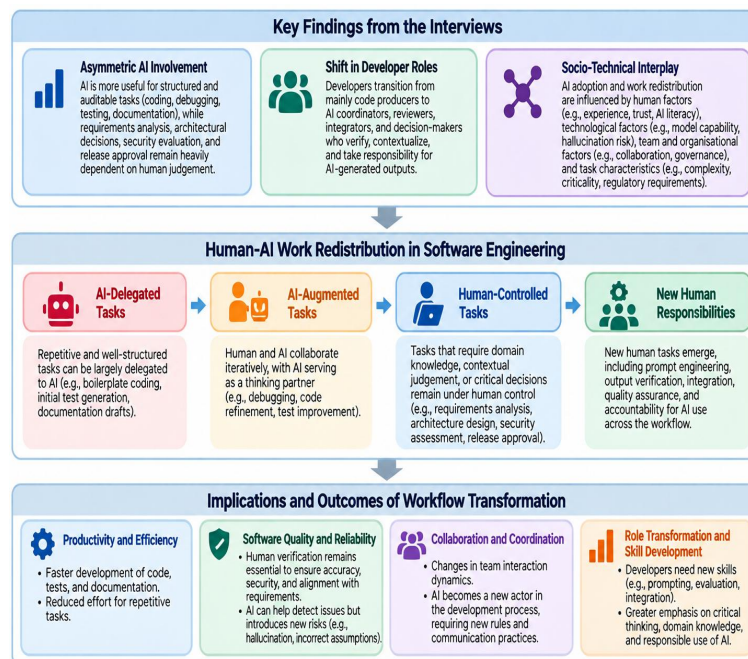


Fig. 3. Empirically derived human-AI work redistribution model for software engineering workflow transformation.

on AI-assisted programming, developer productivity, software testing, and the broader application of LLMs in software engineering [5], [2], [7], [30]. This study contributes qualitative evidence on how software professionals in Indonesia experience the redistribution of work across multiple SDLC activities. However, the patterns identified here should not be interpreted as unique to Indonesian software development. Rather, the Indonesian setting provides a specific empirical context in which we observed these patterns. Comparative studies involving developers from different countries, organizational settings, and software domains are needed to determine which patterns are broadly shared and which are influenced by regional, professional, or organizational conditions.

Participants' productivity benefits also require careful interpretation. In this study, productivity reflects developers' perceptions of changes in their work rather than objectively measured improvements in development performance. Study [2], for example, provides experimental evidence that AI assistance can improve performance in a controlled programming task. The present study addresses a different dimension by examining developers' experiences across a wider range of software engineering activities. These two forms of evidence should therefore not be treated as directly equivalent. The results cannot establish the magnitude of productivity improvement or determine whether task-level efficiencies lead to sustained improvements in project or organizational performance. Future research could address this limitation by combining qualitative evidence with repository analytics, development telemetry, software quality indicators, and longitudinal performance measures. Such an approach would provide stronger evidence on whether changes in human-AI work distribution ultimately translate into measurable improvements in software development outcomes.

V. CONCLUSION

This study examined how AI coding assistants redistribute work between developers and AI across the SDLC, how this shift influences software engineering workflows and developer roles, and which socio-technical factors shape these changes in the context of Indonesian software development.

For RQ1, the results reveal that human and AI responsibilities vary across SDLC activities. The analysis revealed three patterns: AI-delegated, AI-augmented, and human-controlled work. AI plays a greater role in structured, repetitive, and relatively easy-to-verify tasks, such as generating boilerplate code, drafting documentation, and creating initial test cases. Activities such as debugging, code refinement, and test improvement involve closer interaction between developers and AI. Requirements analysis, architectural decisions, security assessment, and release approval remain largely under human control because they rely on contextual understanding, technical judgment, and professional accountability.

For RQ2, the redistribution of work also changes how

developers participate in the development process. As AI supports more production-oriented tasks, developers increasingly provide context, guide AI interactions, review generated outputs, integrate results, and make final engineering decisions. This shift does not remove developers from the workflow. Instead, it shifts the focus of their effort, moving some work from direct production to coordination, validation, integration, and oversight. Human validation therefore remains essential, particularly when AI-generated outputs must be assessed for accuracy, security, contextual relevance, and alignment with project requirements.

For RQ3, the extent of human-AI work redistribution depends on more than AI tools' technical capabilities. Developer experience, AI literacy, trust, model capability, and output reliability influence how confidently AI is used. Task complexity and the level of contextual judgment required also affect whether work can be delegated to AI or needs greater human involvement. Team practices, security requirements, and organizational policies further shape the incorporation of AI Coding Assistants into development activities. These findings suggest that human-AI work redistribution is best understood as a socio-technical process shaped by interactions among people, technology, tasks, and organizational conditions.

Theoretically, this study draws on socio-technical systems theory and activity theory to explain how AI changes the organization of software engineering work. The socio-technical perspective highlights interactions among technological capabilities, developers, work practices, teams, and organizational conditions. Activity theory helps explain AI Coding Assistants as mediating artifacts that reshape development activities and the division of labor. Building on these perspectives and the empirical findings, the proposed Human-AI Work Redistribution model illustrates how responsibilities can shift between developers and AI while human oversight remains important in activities involving judgment, risk, and accountability.

The findings also have practical implications for organizations adopting AI Coding Assistants. Organizations should incorporate human validation into AI-assisted development practices rather than treating it as an additional or optional activity. Organizations need clear procedures for reviewing AI-generated outputs, protecting sensitive information, addressing security risks, and assigning responsibility for AI-assisted decisions. Developers should also go beyond teaching how to use AI tools and strengthen skills in critical evaluation, contextual reasoning, software engineering fundamentals, security awareness, and responsible decision-making.

The study's findings are limited by its sample size of seven software professionals from Indonesia, focusing on contextual insights rather than statistical generalization. The evidence relies on self-reported experiences, lacking objective data like

repository analytics or productivity measurements. Additionally, the use of various AI Coding Assistants by participants may have influenced their experiences. Recall and social desirability may have influenced responses in this interview-based study. The proposed human-AI work redistribution model is an empirically grounded conceptual model needing further validation, not a universal representation of AI-assisted software development.

Future research could examine whether similar patterns of human-AI work redistribution emerge across larger and more diverse developer populations, organizations, and software domains. Longitudinal and mixed-method studies could connect developers' reported experiences with objective engineering evidence using measures such as DORA metrics, SPACE framework indicators, defect density, security findings, and rework rates. Further studies could also evaluate the relationships proposed in the model and investigate how differences in AI tools, developer roles, task characteristics, and organizational environments shape the distribution of work between developers and AI.

ACKNOWLEDGMENT

The authors would like to thank the software developers who shared their experiences and supporting materials for this research. Their participation enabled this research to examine the practical, organizational, and ethical aspects of AI-assisted software development.

REFERENCES

- [1] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, Oct. 1950, doi: 10.1093/mind/LIX.236.433.
- [2] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with AI assistants?," in *Proc. 2023 ACM SIGSAC Conf. Comput. Commun. Secur. (CCS '23)*, 2023, pp. 2785–2799, doi: 10.1145/3576915.3623157.
- [3] J. T. Liang, C. Yang, and B. A. Myers, "A large-scale survey on the usability of AI programming assistants: Successes and challenges," in *Proceedings of the IEEE/ACM 46th*, 2024, pp. 1–13, doi: 10.1145/3597503.3608128.
- [4] S. Barke, M. B. James, and N. Polikarpova, "Grounded Copilot: How programmers interact with code-generating models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1, pp. 85–111, Apr. 2023, doi: 10.1145/3586030.
- [5] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. Experience: Evaluating the usability of code generation tools powered by large language models," *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pp. 1–7, Apr. 2022, doi: 10.1145/3491101.3519665.
- [6] X. Hou et al., "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–79, Sep. 2024, doi: 10.1145/3695988.
- [7] I. Ozkaya, "Application of large language models to software engineering tasks: Opportunities, risks, and implications," *IEEE Softw.*, vol. 40, no. 3, pp. 4–8, 2023, doi: 10.1109/MS.2023.3248401.
- [8] D. Russo, "Navigating the complexity of generative AI adoption in software engineering," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, pp. 1–50, Mar. 2024, doi: 10.1145/3652154.
- [9] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 754–768, May 2022, doi: 10.1109/sp46214.2022.9833571.
- [10] Q. Zhang et al., "A survey on large language models for software engineering," *Science China Information Sciences*, vol. 69, no. 4, Mar. 2026, doi: 10.1007/s11432-025-4670-0.
- [11] S. S. M. Sinaga, Z. Indra, M. R. Wijaya, and M. G. Almahdi, "The role of generative AI in the software development life cycle: A systematic literature review," *Journal of Artificial Intelligence and Engineering Applications (JAIEA)*, vol. 5, no. 3, pp. 4273–4278, Jun. 2026, doi: 10.59934/jaiea.v5i3.2431.
- [12] A. Sergeyuk, Y. Golubev, T. Bryksin, and I. Ahmed, "Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward," *Information and Software Technology*, vol. 178, Art. no. 107610, Oct. 2024, doi: 10.1016/j.infsof.2024.107610.
- [13] A. Mastropaolo et al., "On the robustness of code generation techniques: An empirical study on GitHub Copilot," in *Proceedings - International Conference on Software Engineering*, 2023, pp. 2149–2160, doi: 10.1109/ICSE48619.2023.00181.
- [14] G. Baxter and I. Sommerville, "Socio-technical systems: From design methods to systems engineering," *Interact. Comput.*, vol. 23, no. 1, pp. 4–17, 2011, doi: 10.1016/j.intcom.2010.07.003.
- [15] E. Mumford, "The story of socio-technical design: Reflections on its successes and failures," *Inf. Syst. J.*, vol. 16, no. 4, pp. 317–342, 2006, doi: 10.1111/j.1365-2575.2006.00221.x.
- [16] M. Maguire, "Socio-technical systems and interaction design – 21st century relevance," *Appl. Ergon.*, vol. 45, no. 2, pp. 162–170, 2014, doi: 10.1016/j.apergo.2013.05.011.
- [17] C. Clegg, "Sociotechnical principles for system design," *Appl. Ergon.*, vol. 31, no. 5, pp. 463–477, 2000, doi: 10.1016/S0003-6870(00)00009-0.
- [18] Y. Engström, *Learning by Expanding: An Activity-Theoretical Approach to Developmental Research*. Helsinki, Finland: Orienta-Konsultit, 1987.
- [19] V. K. and B. A. Nardi, "Acting with Technology: Activity Theory and Interaction Design," in *First Monday*, Cambridge, MA, USA: MIT Press, 2006, doi: 10.5210/fm.v12i4.1772.
- [20] D. Mwanza, "Towards an activity-oriented design method for HCI research and practice," *Open Research Online (the Open University)*, Jan. 2002, doi: 10.21954/ou.ro.0000dc95.
- [21] V. Kaptelinin, "Activity theory as a framework for human-computer interaction research," *Mind, Cult. Act.*, vol. 17, no. 1, pp. 1–17, 2010, doi: 10.1080/10749039.2017.1393089.
- [22] I. Seeber et al., "Machines as teammates: A research agenda on AI in team collaboration," *Information & Management*, vol. 57, no. 2, Art. no. 103174, Jul. 2019, doi: 10.1016/j.im.2019.103174.
- [23] R. P. and C. A. Riley, "Humans and automation: Use, misuse, disuse, abuse," *Hum. Factors*, vol. 39, no. 2, pp. 230–253, 1997, doi: 10.1518/001872097778543886.
- [24] J. D. L. and K. A. See, "Trust in automation: Designing for appropriate reliance," *Hum. Factors*, vol. 46, no. 1, pp. 50–80, 2004, doi: 10.1518/hfes.46.1.50.30392.
- [25] B. J. Dietvorst, J. P. Simmons, and C. Massey, "Algorithm aversion: People erroneously avoid algorithms after seeing them err.," *Journal of Experimental Psychology General*, vol. 144, no. 1, pp. 114–126, Nov. 2014, doi: 10.1037/xge0000033.
- [26] V. B. and V. Clarke, "Using thematic analysis in psychology," *Qual. Res. Psychol.*, vol. 3, no. 2, pp. 77–101, 2006, doi: 10.1191/1478088706qp063oa.
- [27] V. B. and V. Clarke, "Reflecting on reflexive thematic analysis," *Qual. Res. Sport. Exerc. Heal.*, vol. 11, no. 4, pp. 589–597, 2019, doi: 10.1080/2159676X.2019.1628806.
- [28] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of AI on developer productivity: Evidence from GitHub Copilot," *arXiv (Cornell University)*, Feb. 2023, doi: 10.48550/arxiv.2302.06590.
- [29] R. Hoda, H. Dam, C. Tantithamthavorn, P. Thongtanunam, and M.-A. Storey, "Augmented Agile: Human-centered AI-assisted software management," *IEEE Software*, vol. 40, no. 4, pp. 106–109, Jul. 2023, doi: 10.1109/ms.2023.3268725.
- [30] X. Hou et al., "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–79, Sep. 2024, doi: 10.1145/3695988.
- [31] F. Wierdmann, M. Borg, N. Hagatulah, and A. Tornhill, "Code health in LLM-based test generation: effectiveness and token efficiency," *arXiv (Cornell University)*, Aug. 2026, doi: 10.48550/arxiv.2608.18645.
- [32] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, Jan. 2006, doi: 10.1191/1478088706qp063oa.