Generative AI in Software Engineering Education: A Study on Competency and Perception

Ghozy Ghulamul Afif*
Faculty of Computer Science
Universitas Indonesia
Depok, Indonesia
ghozy.ghulamul@ui.ac.id

Panca Oktavia Hadi Putra
Faculty of Computer Science
Universitas Indonesia
Depok, Indonesia
hadiputra@cs.ui.ac.id

Abstract

Software Engineering (SE) education struggles to align graduate competencies with industry needs, a challenge compounded by Generative AI (GenAI). This study's purpose is to explore how structured GenAI utilization can enhance student competencies and learning perceptions. This exploratory case study used a mixed-method design involving four students in a controlled experiment. Data was collected via quantitative (SWEBOK-based artifact assessments, questionnaires) and qualitative narrative reflections. Results show the group using AI produced higher-quality artifacts and reported a more efficient, positive experience. A key finding is the transformation of work processes, shifting the primary challenge from manual execution to the critical evaluation of AI outputs. Due to its small scale, the findings are preliminary but suggest GenAI can enhance competencies. This requires an educational shift toward human-AI collaboration skills like critical evaluation and prompt engineering. Further validation with larger, more comprehensive studies is essential to generalize these initial insights.

Keywords: Generative AI, software engineering education, skills gap, SWEBOK, case study, human-AI collaboration, prompt engineering

Introduction

The global technology sector is navigating a period of notable adjustment. Recent trends, including widespread layoffs during the "tech winter," have created a complex job market (Afifa, 2023). Paradoxically, this volatility has only intensified a persistent skills gap, with a majority of business executives now predicting a significant talent shortage (Cantrell, 2022). This misalignment between the competencies of university graduates and the dynamic needs of the software industry has been a documented challenge in Software Engineering (SE) education for decades. Educators continuously struggle to develop relevant competencies—such as requirements analysis, architectural design, and complex problem-solving—that extend beyond mere coding proficiency (Garousi et al., 2020; Oguz & Oguz, 2019; Zönnchen et al., 2024).

This foundational challenge is now being reshaped by the emergence of Generative Artificial Intelligence (GenAI). The rapid adoption of GenAI tools in the industry is transforming software development workflows, with studies reporting significant productivity gains and a shift in the nature of developer tasks (Coutinho et al., 2024; Gao, 2024). This technological wave presents both a challenge

* Corresponding Author

and an opportunity for SE education. It presents challenges to traditional pedagogical approaches while also offering a powerful means to enhance learning, for instance, by helping students brainstorm requirements (Pereira et al., 2024), generate initial code (Zönnchen et al., 2024), or understand complex design concepts (Chen, 2024). It is important to move beyond speculation and empirically investigate how these tools can be leveraged effectively.

While the potential of GenAI in education is a topic of widespread discussion, there remains a notable gap in systematic, evidence-based research on its structured application within the SE curriculum. Current educational approaches face hurdles in keeping curricula relevant to rapid technological change (Kozov et al., 2024) and in providing scalable, personalized feedback to students (Sengul et al., 2024). Many existing studies on GenAI in this context are exploratory, focus on user perceptions, or are limited to basic programming tasks (Sengul et al., 2024). It is still largely unclear how the integration of GenAI as a core component of the development process—rather than just an auxiliary tool—affects specific student competencies as defined by established frameworks like the Software Engineering Body of Knowledge (SWEBOK).

This discrepancy between the rapid adoption of GenAI in industry and its cautious, often unstructured, application in higher education creates a notable tension. While industry professionals are leveraging these tools to accelerate development, academic institutions are left without clear, evidence-based guidance on how to prepare the next generation of software engineers for this new reality. This study addresses this research gap by examining the direct impact of structured GenAI use on student performance and learning experiences in a controlled educational setting. Therefore, the primary objective of this research is to explore how the structured integration of GenAI into software engineering coursework influences student competency development and their perceptions of the learning process.

To achieve this objective, this study is guided by the following central research question: *"How can the utilization of Generative AI in the Software Engineering learning process enhance the competencies of IT students to meet industry needs?"* By focusing on this question, the research aims not to provide a comprehensive solution to the entire skills gap, but to investigate the potential of GenAI as a strategic tool to address specific facets of it, such as accelerating the mastery of fundamental SE skills and exposing students to the latest technologies being adopted by the industry.

This study is expected to provide contributions to both theory and practice. Theoretically, it will offer a preliminary framework for understanding the role of human-AI collaboration in competency development within SE education. This aligns with a growing body of research indicating an evolving skill shift away from manual coding and toward strategic competencies like critical output evaluation and prompt engineering (Fan et al., 2023; Ozkaya & Schmidt, 2024; Zönnchen et al., 2024). Practically, the findings will provide educators and curriculum designers with initial, evidence-based insights into how they can effectively adapt their teaching methodologies for the GenAI era. By illuminating the benefits, challenges, and the "learning paradox"—where effective AI use requires strong foundational knowledge—this research aims to help shape a pedagogical approach that prepares students for the future of software engineering.

Literature Review

Definition and Fundamental Principles of SE

Software Engineering (SE) is formally defined as the application of a systematic, disciplined, and measurable approach to the development, operation, and maintenance of software (International Organization for Standardization, 2011). This discipline, as articulated by pioneers like Fritz Bauer and Ian Sommerville, emphasizes producing economically efficient and technically reliable software that encompasses all aspects of its production lifecycle (Bauer, 1972; Sommerville, 2011). Despite its status, SE is an evolving field, and unlike traditional engineering disciplines, it has not yet reached a universal consensus on its fundamental principles, reflecting an ongoing refinement of its theoretical foundations (Al-Sarayreh et al., 2021).

SWEBOK: The Map of Software Engineering Knowledge

A significant step toward standardizing competencies and knowledge in SE is the development of the Software Engineering Body of Knowledge (SWEBOK). Compiled as an official guide by the IEEE Computer Society and adopted as ISO/IEC TR 19759, SWEBOK establishes a generally accepted knowledge framework for the SE community. Its primary goal is to foster a uniform understanding of the discipline globally and to serve as a basis for educational curricula and professional certification (Washizaki et al., 2025).

Table 1. SWEBOK v4.0 KAs (Washizaki, 2024)

No	Knowledge Areas
1	Software Requirements
2	Software Architecture
3	Software Design
4	Software Construction
5	Software Testing
6	Software Engineering Operations
7	Software Maintenance
8	Software Configuration Management
9	Software Engineering Management
10	Software Engineering Process
11	Software Engineering Models and Methods
12	Software Quality
13	Software Security
14	Software Engineering Professional Practice
15	Software Engineering Economics
16	Computing Foundations
17	Mathematical Foundations
18	Engineering Foundations

The latest edition, SWEBOK v4.0 released in 2024, organizes the SE discipline into 18 main Knowledge Areas (KAs) that cover the entire software lifecycle and relevant engineering practices. These KAs range from requirements analysis to software maintenance, including managerial and quality assurance aspects, underscoring that SE is a complex and comprehensive discipline that extends far beyond coding. Table 1 lists these 18 Knowledge Areas.

Competency Gap

SE education aims to prepare students for industry careers by delivering the conceptual foundations of the software lifecycle and developing professional competencies through practical experience (Kirova et al., 2024). However, a persistent gap exists between the competencies of IT graduates and the actual needs of the software industry (Garousi et al., 2020; Oguz & Oguz, 2019). This gap is often attributed to factors such as curricula that are not aligned with industry dynamics, insufficient student experience in real-world projects, and suboptimal development of soft skills during the educational process (Garousi et al., 2020; Ozkaya & Schmidt, 2024).

SE competency itself is a crucial combination of knowledge, skills, and personal attributes required by the industry for effective software development (Assyne et al., 2022b, 2022a; Rivera-Ibarra et al., 2010). This includes technical mastery aligned with SWEBOK knowledge areas and essential soft skills like

problem-solving, communication, and teamwork (Assyne et al., 2022a; Garousi et al., 2020; Kirova et al., 2024). The rise of GenAI is predicted to shift competency needs, demanding that professionals be able to analyze, validate, and integrate AI-generated work meticulously (Pereira et al., 2024; Zönnchen et al., 2024). The focus may shift from manual coding to higher-level tasks such as design, requirements engineering, and testing of AI outputs (Zönnchen et al., 2024). This shift adds another layer of complexity to the challenge of bridging the existing skills gap, underscoring the urgency for curriculum adaptation in SE education (Ozkaya & Schmidt, 2024).

Generative Artificial Intelligence (GenAI)

Generative Artificial Intelligence (GenAI) is technology capable of producing new and original content such as code, designs, or documentation in Software Engineering (SE) (Pereira et al., 2024). This technology is being adopted to increase productivity and simplify the development process (Coutinho et al., 2024), with applications spanning various stages from requirements elicitation, diagram creation, and API simulation to code writing, explanation, testing, and documentation (Pereira et al., 2024).

While tools like GitHub Copilot and ChatGPT are often used to support these activities (Coutinho et al., 2024), the effectiveness of their use heavily depends on user interaction, where careful prompt design (Pereira et al., 2024) and the ability to formulate precise instructions are key to obtaining desired results (Coutinho et al., 2024). The generated outputs often require manual refinement and critical validation by the user (Coutinho et al., 2024; Kuhail et al., 2024; Pereira et al., 2024) due to significant challenges. These challenges include issues of reliability and accuracy (Coutinho et al., 2024; Pereira et al., 2024), the potential for hallucination or generating misleading information (Pereira et al., 2024), the risk of over-reliance that can hinder skill development (Pereira et al., 2024), potential biases in the models (Pereira et al., 2024), and concerns related to security and privacy when handling sensitive data (Coutinho et al., 2024). Therefore, understanding both the potential and the challenges, along with the importance of skilled interaction, is essential for the effective use of GenAI in SE.

Methodology

Research Design

This study employs an explanatory mixed-method design, which is well-suited for research that aims to first identify quantitative outcomes and then explain those results with in-depth qualitative data (Creswell, 2014). The research was structured as an evaluative case study involving an experimental design. Four third-year (6th semester) IT students were selected through purposive sampling and divided into two groups: a control group (manual work) and an experimental group (using Generative AI).

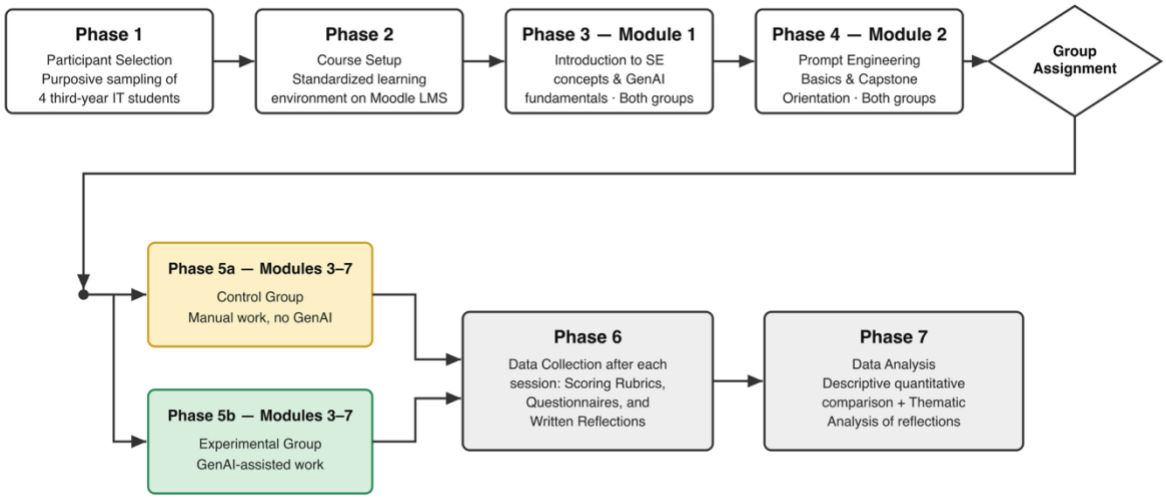


Figure 1. Research Process Flowchart

Figure 1 illustrates the overall research process from beginning to end. The study was organized into seven phases. It began with participant selection and the assignment of students to control and experimental groups, followed by the preparation of the course on the Moodle LMS. In Module 1, both groups were introduced to fundamental concepts in Software Engineering (SE) and Generative AI (GenAI). Module 2 focused on prompt engineering and provided guidance for the capstone project.

The main project activities were conducted across five sessions (Modules 3–7). During these sessions, the control group completed the tasks manually, while the experimental group was allowed to use GenAI tools. Each session was aligned with a specific SWEBOK Knowledge Area. After every session, data were collected using scoring rubrics, questionnaires, and students' written reflections. The final stage consisted of a two-phase analysis that combined descriptive quantitative comparisons with thematic analysis of the qualitative data. The figure also indicates the stage where the two groups began to differ, starting from Module 3, and the point where their data were brought together again for the final analysis.

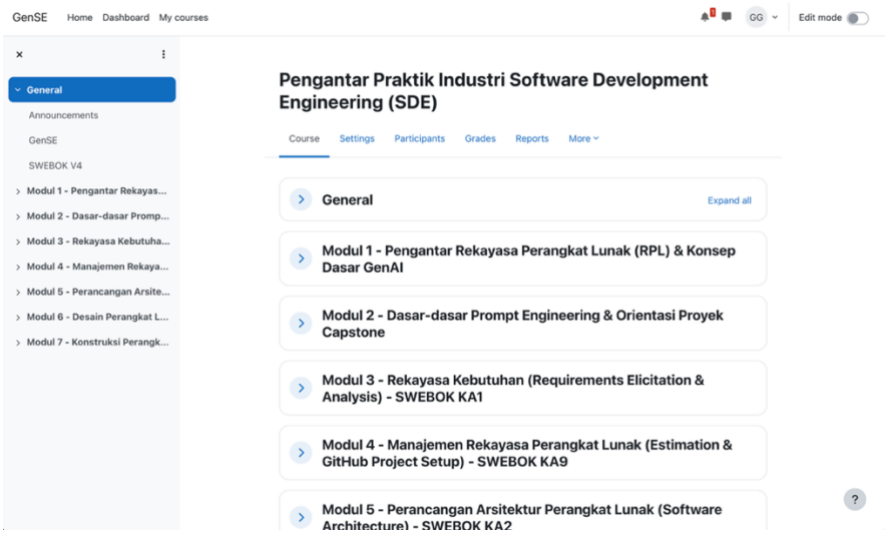


Figure 2. Structured Learning Environment on Moodle

The independent variable was the use of GenAI, while the dependent variables were the quality of the software engineering artifacts produced and the students' perceptions of the learning experience. The entire experiment was facilitated through a dedicated course on the Moodle Learning Management System (LMS) to ensure standardized delivery of materials, instructions, and data collection across both groups. The research process followed a sequential structure, beginning with two preparatory modules and followed by five core project sessions aligned with key SWEBOK Knowledge Areas: Software Requirements, SE Management, Software Architecture, Software Design, and Software Construction. Figure 2 shows the resulting structured learning environment on Moodle.

This study is positioned as an exploratory, education-oriented case study that aims to identify emerging patterns in students' learning processes and competency development when Generative AI is integrated into Software Engineering coursework. Given the small sample size and the pedagogical focus of the research, the study does not seek statistical generalization. Instead, it emphasizes in-depth analysis of observed tendencies, supported by descriptive quantitative results and qualitative reflections, in line with established practices in computing education research.

Research Instruments

To align with the mixed-method research design, both quantitative and qualitative data collection instruments were employed as follows:

1. **Scoring Rubrics**

   Structured rubrics were developed before data collection began for each of the five project sessions. Each rubric translated the relevant SWEBOK competency into a set of specific criteria

rated on a five-point scale, with distinct behavioral descriptors defined for every score level. The criteria addressed aspects such as the completeness and testability of acceptance criteria, the quality of architectural justification, and the functional correctness of the implemented code.

All artifacts were scored by the primary researcher. The researcher holds a graduate degree in Computer Science with a focus on Software Engineering and has practical experience in software development across the knowledge areas being assessed. Only one scorer was involved, which is acknowledged as a limitation later in this paper. To maintain consistency in scoring, three measures were applied. First, the rubric descriptors were finalized before any student work was reviewed so that the evaluation criteria remained fixed throughout the study. Second, a written justification was recorded for each assigned score at the time of assessment, providing a clear rationale for every evaluation decision. Third, each artifact was assessed against the rubric criteria independently rather than through comparison with other participants' work. Because the study used a single scorer, no inter-rater reliability coefficient was calculated. This limitation is discussed further in a later section.

2. **Learning Experience Questionnaire**

   A Likert-scale questionnaire was administered after each session to gather quantitative data on students' subjective learning experiences. The instrument captured responses related to perceived task difficulty, confidence in the quality of their outputs, satisfaction with the development process, perceived efficiency, and comprehension of key software engineering concepts.

3. **Written Reflections**

   To obtain qualitative insights, participants were invited to complete a guided reflection consisting of open-ended questions after each session. These reflections solicited narrative accounts regarding the challenges encountered, problem-solving strategies adopted, and critical takeaways from the learning experience. The qualitative data served to contextualize and deepen the interpretation of the quantitative results. Figure 3 presents the written reflection prompt used for this data collection.



Figure 3. Written Reflection Prompt for Data Collection

Data Collection

Data collection was conducted systematically through the Moodle platform following the completion of each of the five core project sessions (Session 3 to Session 7). This centralized approach ensured that all data was collected in a standardized manner.

The quantitative data collection involved two primary sources. First, the software engineering artifacts (e.g., Product Requirements Document, architecture diagrams, source code) produced by each student

were uploaded to designated submission slots on Moodle. Second, responses to the learning experience questionnaire were collected directly through the platform's questionnaire feature at the end of each session.

The qualitative data were gathered from the written narrative reflections submitted by participants through Moodle. This method allowed for the consistent collection of in-depth insights into each student's thought process and experience throughout the project.

Data Analysis

The data analysis was conducted in two phases that together reflect the explanatory mixed-method design of this study. In the first phase, the quantitative data, consisting of artifact scores and questionnaire responses, were examined. An important consideration is that each group included only two participants. Because of this small sample, applying standard inferential statistical tests would be inappropriate. For this reason, the analysis relies on descriptive comparison rather than hypothesis testing. Each participant's scores were averaged across the five sessions to produce a single aggregate value per person, and these values were then compared between groups. To provide a standardized indication of the magnitude of the observed difference, Hedges' g was calculated using participant-level mean scores ($g \approx 3.48$), indicating a very large descriptive separation between groups. However, given the extremely small sample size ($n = 2$ per group), this value should be interpreted only as an exploratory magnitude indicator rather than a stable estimate of population effect size. Accordingly, no p-values are reported, because inferential testing would require treating repeated session scores as independent observations, which would violate the assumption of independence and risk pseudoreplication.

In the second phase, the written reflections were analyzed using Thematic Analysis. The responses were reviewed, coded, and grouped into recurring patterns that captured how students in each group experienced the project, including the challenges they encountered, the strategies they used, and the lessons they drew from each session. The resulting themes were then used to help interpret the quantitative findings from the first phase, linking the numerical patterns in the scores to the underlying learning experiences that may have produced them.

Result

Descriptive Overview of Group Performance

Before discussing the results by knowledge area, a brief note on interpretation is necessary. With only two participants per group, none of the comparisons here should be read as statistically conclusive. The numbers are reported because they reveal a consistent directional pattern worth examining, but readers should treat them as preliminary observations from a small exploratory study rather than evidence that generalizes beyond these four students.

When each participant's scores are averaged across the five sessions, P3 achieved a mean of 93.0, and P4 achieved 89.0, while P1 and P2 scored 74.0 and 69.0, respectively. Both experimental group members thus outperformed both control group members on this aggregate measure. Hedges' g computed from these four participant means was 3.48, suggesting a large observed difference (Hedges, 1981). However, with $n = 2$ per group, this value should be interpreted as descriptive only. Table 2 breaks down the scores session by session.

Table 3 presents each participant's average questionnaire score for each dimension across the five sessions (scale 1–5). A consistent pattern is evident. On every dimension, both P3 and P4 rated their experience higher than P1 and P2. The gap in completion time was also notable. Because P1's time for the Software Construction session was not recorded, time comparisons are based on the four sessions where complete data exists for all participants. On this basis, the control group averaged approximately 260 minutes per session (P1: 290 min; P2: 230 min), compared to around 76 minutes for the experimental group (P3: 76 min; P4: 76 min).

Table 2. Artifact Scores by Knowledge Area and Participant

Knowledge Area	P1 (Control)	P2 (Control)	Control Avg	P3 (Exp.)	P4 (Exp.)	Exp. Avg	Difference
Software Requirements	100	80	90.0	100	100	100.0	+10.0
SE Management	70	90	80.0	100	70	85.0	+5.0
Software Architecture	80	30	55.0	80	80	80.0	+25.0
Software Design	90	65	77.5	95	95	95.0	+17.5
Software Construction	30	80	55.0	90	100	95.0	+40.0
Overall Mean	**74.0**	**69.0**	**71.5**	**93.0**	**89.0**	**91.0**	**+19.5**

Table 3. Per-Participant Average Questionnaire Scores (Scale 1–5)

Dimension	P1 (Control)	P2 (Control)	Control Avg	P3 (Exp.)	P4 (Exp.)	Exp. Avg
Perceived Task Difficulty	1.4	1.8	1.6	3.4	3.8	3.6
Confidence in Output Quality	1.0	1.8	1.4	4.4	4.2	4.3
Process Satisfaction	2.4	1.8	2.1	4.4	4.2	4.3
Process Efficiency	1.6	1.8	1.7	4.2	4.6	4.4
Core Concept Understanding	2.2	1.8	2.0	3.8	4.4	4.1

Taken together, the descriptive data points in the same direction across both instruments. The experimental group produced higher-quality artifacts and reported a more positive learning experience throughout the sessions. The difference was modest in the earlier, more structured activities and became more pronounced as the tasks grew more technically demanding, particularly in Software Architecture, Software Design, and Software Construction. The sections below examine each knowledge area in detail, drawing on the qualitative reflections to explain what drove these patterns.

Artifact Quality and Performance

The primary quantitative findings are derived from the descriptive comparison of software engineering artifact quality across five SWEBOK Knowledge Areas (KAs). An aggregate review of rubric scores reveals a consistent tendency for participants in the GenAI-supported group to produce artifacts with higher average scores across most assessed competencies compared to the control group. These results reflect observable patterns rather than statistically significant differences and are interpreted within the exploratory scope of the study. Figure 4 compares the competency profiles of the individual participants.

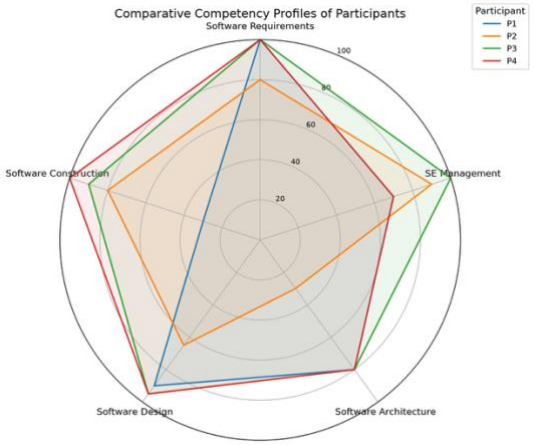


Figure 4. Comparative Competency Profiles of Participants

A breakdown of performance by each SWEBOK Knowledge Area provides more specific insights, integrating qualitative reflections to explain the quantitative scores. Figure 5 summarizes the average artifact quality scores by knowledge area.

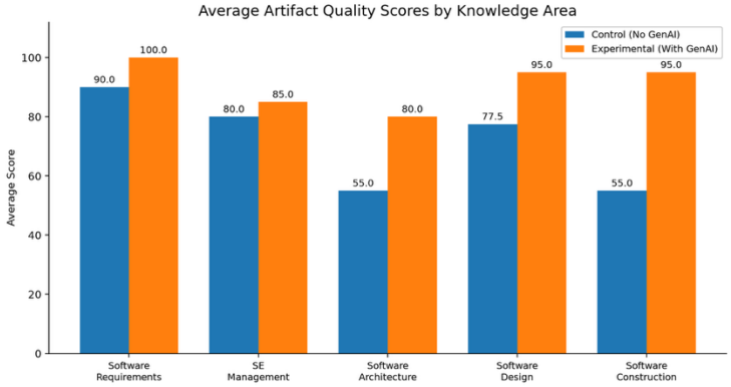


Figure 5. Average Artifact Quality Scores by Knowledge Area

1. **Software Requirements (KA1)**

   The experimental group achieved consistently optimal scores, while the control group's performance was varied. This discrepancy is explained by the different strategies employed. The control group relied on manual cognitive effort, with one participant noting the primary challenge was *"...composing clear and unambiguous Acceptance Criteria and Non-Functional Requirements"* (P2, Control). In contrast, the experimental group leveraged GenAI as a core strategy. One participant stated their process was simply *"Using GenAI"* (P4, Experimental), which standardized their output at a high level of quality by assisting with tasks like brainstorming and identifying edge cases.

2. **Software Engineering Management (KA9)**

   Performance in this area was highly heterogeneous, with the GenAI intervention not guaranteeing superior results. The key challenge for both groups was justifying effort estimation, a task requiring experience. The experimental group's reliance on AI for this task proved to be a "black box." One participant (P3, Experimental) admitted, *"...even though GPT gave an estimate, I couldn't justify it,"* while another (P4, Experimental) stated, *"The determining factors: GenAI decided them here."* This lack of personal rationale explains why their estimation scores were lower than a high-performing control group participant who relied on experience.

3. **Software Architecture (KA2)**

   The results were divergent. The experimental group excelled in providing detailed justifications for technology choices, using AI as a research assistant to compare options. However, a control group participant (P1) produced the highest quality architectural diagram. The experimental group's reflections reveal the limitations of AI in this visual task, with one noting, *"...using ChatGPT to help make the diagram—the format was often inconsistent..."* (P3, Experimental). This suggests that for tasks requiring formal visual abstraction, individual capability remains dominant.

4. **Software Design (KA3)**

   The experimental group showed a slight overall superiority, driven by notably higher scores in creating Sequence Diagrams. The qualitative data reveal a fundamental difference like the challenge. The control group experienced a high cognitive load, which one participant described as *"traumatizing"* (P1, Control). The experimental group, however, faced a different challenge: evaluation. Their advantage in creating Sequence Diagrams stemmed from an AI-driven reverse-engineering strategy, where they prompted the AI to analyze existing code to generate the diagram automatically.

5. **Software Construction (KA4)**

   A clear performance hierarchy emerged, with the experimental group demonstrating markedly superior outcomes. GenAI acted as a powerful accelerator for participants who already possessed foundational skills. This was articulated by one participant (P3, Experimental) who noted, *"...the development process using LLM was faster, especially... the Copilot agent could read the code structure."* This allowed them to focus on business logic rather than boilerplate code. Conversely, for participants with weaker foundational skills, the time saved on writing code was lost in the effort required to integrate and revise the AI's output, diminishing the tool's effectiveness.

Student Perceptions and Learning Experience

Data from the post-session questionnaires revealed notable differences in the subjective learning experience between the two groups across several dimensions.

1. **Time and Efficiency**

   The experimental group consistently reported requiring substantially less time to complete tasks. For instance, in the SE Management session, the control group spent an average of 245 minutes, whereas the experimental group spent only 50 minutes. This trend was consistent across all sessions. This pattern is illustrated in Figure 6 and Figure 7.

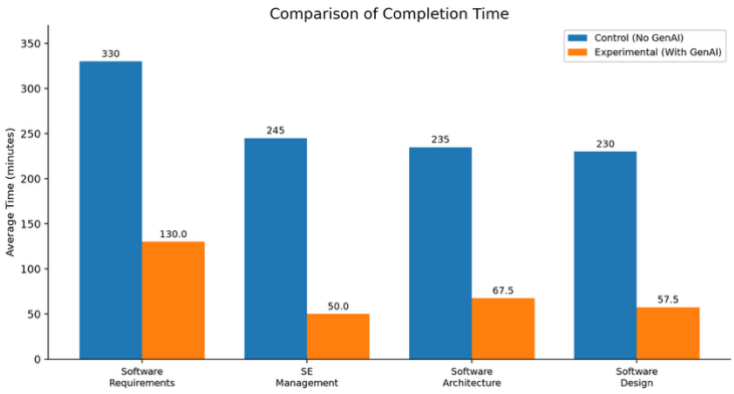


Figure 6. Comparison of Completion Time

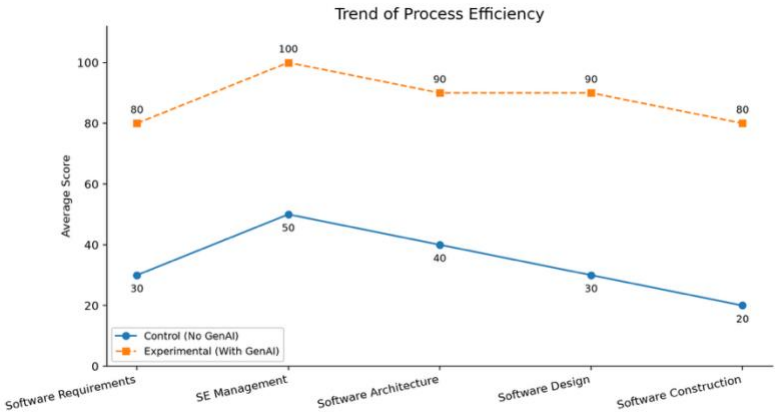


Figure 7. Trend of Process Efficiency

2. Confidence and Difficulty

Participants using GenAI reported substantially higher confidence in the quality of their work and perceived the tasks as less difficult compared to the control group. These trends are shown in Figure 8 and Figure 9.

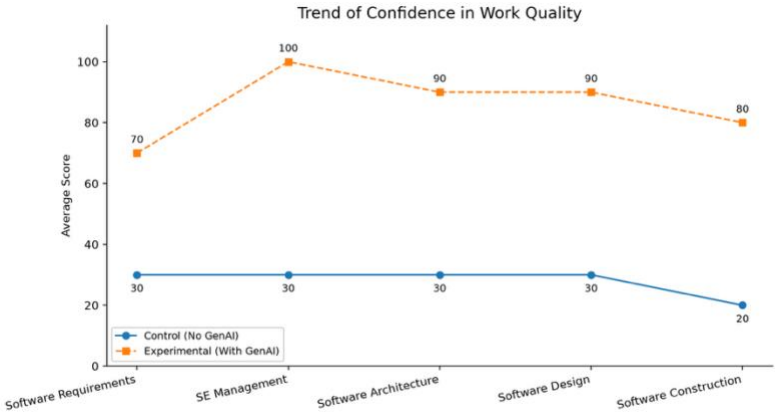


Figure 8. Trend of Confidence in Work Quality

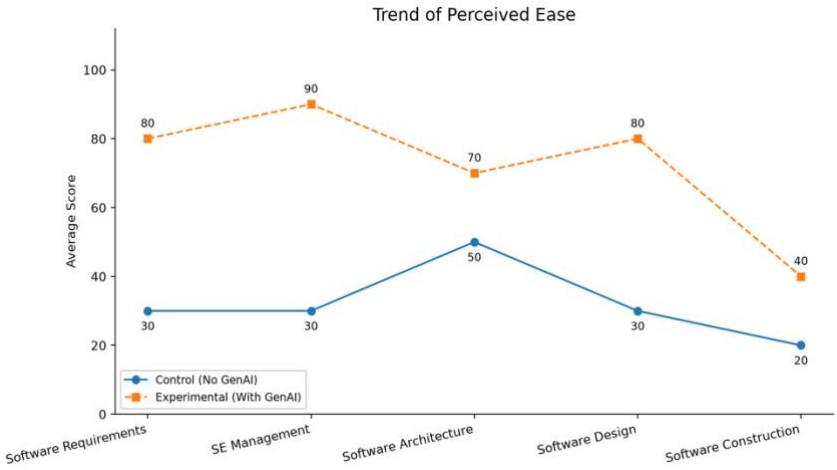


Figure 9. Trend of Perceived Ease

3. Concept Comprehension

The experimental group consistently reported a high level of understanding of the core concepts for each session. The control group's self-reported understanding was lower and more variable, particularly in the more complex design and construction phases. Figure 10 depicts this trend in core concept understanding.

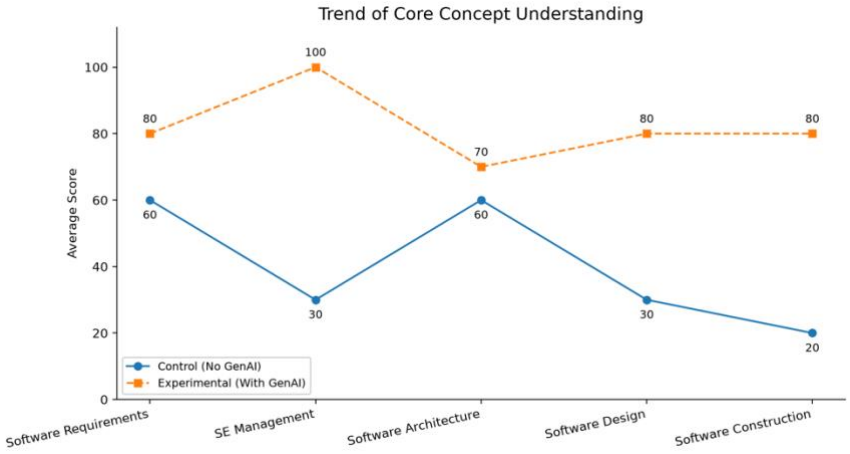


Figure 10. Trend of Core Concept Understanding

Theme 1: Paradigm Shift in Work Strategy: Manual vs. AI-Orchestrated

A fundamental difference in work strategy was evident between the groups. The control group relied on manual cognitive processes, involving independent research and analogical reasoning to complete tasks. This placed the entire analytical burden on the individual.

"...looking at examples... searching for precedents from similar applications... [doing] additional research..." - P1 (Control)

In contrast, the experimental group integrated GenAI as a core component of their workflow, transforming the process from manual execution to AI-assisted orchestration. This advanced strategy included using AI for reverse engineering, which helps explain the clear descriptive performance difference observed in the quantitative scores for complex tasks like Software Design and Construction.

"...asked the agent copilot to access the source code and create an ERD, Sequence Diagram... based on the source code..." - P4 (Experimental)

Theme 2: Transformation of Challenges: From Burden of Execution to Burden of Evaluation

The nature of the primary challenge faced by the participants was fundamentally different. The control group struggled with the burden of execution—the high cognitive load required to create complex artifacts from scratch. This was acutely felt in the design phase.

"[The design task was] traumatizing because of the high complexity and precision required in the manual process." - P1 (Control)

Conversely, the experimental group's main challenge shifted to the burden of evaluation. Their difficulty was not in generating content, but in critically assessing, contextualizing, and validating the output provided by the AI. This explains why their performance was not uniformly perfect, particularly in the SE Management task, where analytical justification was paramount.

"...creating the diagram... wasn't hard... the hard part was having to evaluate the results/answers from GenAI." - P4 (Experimental)

Theme 3: The Learning Paradox: Efficiency vs. Depth of Understanding

A recurring theme was the perceived trade-off between the efficiency gained from using AI and the depth of understanding acquired. Participants acknowledged that a manual process, while slower, often felt like it led to a more profound comprehension. However, they also realized that using AI effectively was not a shortcut that bypassed the need for knowledge. Instead, a strong foundational understanding was a prerequisite to guide the AI and validate its output, preventing it from becoming a passive user.

"...if we can't understand the code, we will surely be easily replaced by GenAI in the future." - P4 (Experimental)

Theme 4: Redefinition of Engineer Competencies

Across their reflections, participants described a shift in how they understood their role as engineers, moving away from the view of a developer as someone who manually produces outputs toward a more evaluative and coordinating role. Figure 11 presents the four competencies that emerged from this theme and illustrates how they relate to one another. The arrows are not merely decorative. They represent a working cycle. When working with AI, an engineer first provides context, since without it the AI tends to generate generic and often unusable outputs. This context then informs the prompt, introducing the second competency, prompt design. The AI subsequently generates an output, after which the third competency, critical validation, becomes necessary to review, correct, and determine what should be retained. Overarching these steps is the fourth role, ethical guardianship, which functions less as a separate stage and more as an ongoing responsibility that influences decisions throughout the cycle. The arrows between the components are bidirectional to indicate that the process is iterative. If validation fails, the engineer may return to refine the prompt or reconsider the context until the output meets the required standard.

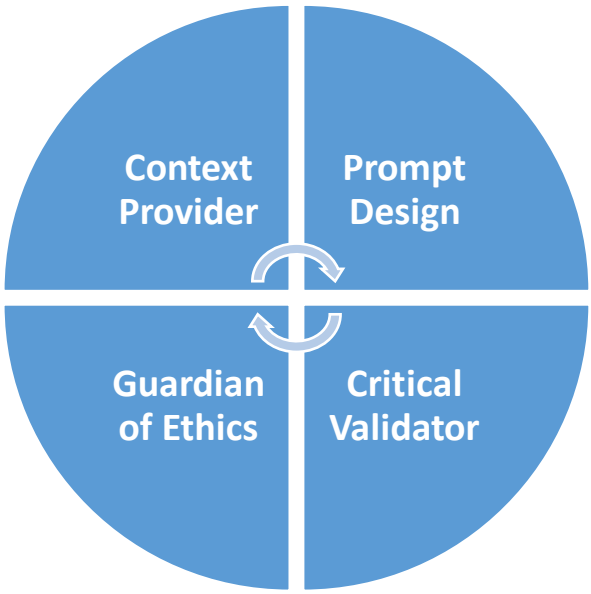


Figure 11. Four Key Competencies of Software Engineers in the AI Era

Each competency is elaborated below, supported by participants' own reflections:

1. **Context Provider**

   Successful interaction with AI was found to be highly dependent on providing precise and complete problem context. Participants realized that without it, AI tends to produce generic or

unsuitable output. As one participant concluded, *"...the thinking must still come from the programmer, because the [AI's] context is limited"* (P4, Experimental).

2. **Prompt Design**

   The ability to design effective and clear commands, or prompts, was recognized as key to obtaining desired results from the AI. One participant explicitly mentioned the challenge of *"...composing a clear prompt..."* as a primary difficulty when interacting with the LLM.

3. **Critical Validator**

   The first and most fundamental competency is the ability to not accept AI output at face value, but to actively validate, test, and refine it. This was described as the most difficult part of the process, with one participant stating, *"...the hard part is having to evaluate the results/answers from GenAI..."* (P4, Experimental). This competency shifts the engineer's role from "creator" to "curator" and "quality assurer".

4. **Guardian of Ethics**

   Finally, participants recognized a broader role that includes the responsibility to reshape *"...ethical standards..."* in using AI technology responsibly. These four pillars collectively redefine the profile of a software engineer who is relevant and ready to face the challenges of the modern industry.

Discussion and Implications

This study demonstrates that integrating Generative AI (GenAI) into software engineering coursework fundamentally shifts students' focus from routine artifact creation toward critical evaluation and higher-order design activities. Rather than simply automating coding or modeling tasks, GenAI tools serve as catalysts for deeper engagement: students allocate saved time to validate AI outputs, refine architectural decisions, and articulate rationale for their design choices (Fan et al., 2023).

A key novel insight is the "learning paradox": while GenAI accelerates production of deliverables, it simultaneously demands stronger conceptual understanding to detect and correct AI-introduced inaccuracies. In educational settings, this paradox implies that GenAI adoption without scaffolded support may inadvertently mask learning gaps (Zönnchen et al., 2024). Accordingly, curricula should be redesigned to balance tool-driven efficiency with structured opportunities for manual problem solving and error analysis (Pereira et al., 2024).

Practically, educators can leverage these findings by:

1. Embedding prompt engineering exercises that guide students in formulating effective queries and interpreting AI responses (Coutinho et al., 2024).
2. Incorporating reflection checkpoints where learners must critique AI outputs, document identified errors, and propose corrective actions.
3. Adopting hybrid assignments that alternate between fully manual implementations and AI-augmented workflows, ensuring proficiency in both modalities (Ozkaya & Schmidt, 2024).
4. Updating assessment rubrics to evaluate not only artifact quality but also the rigor of students' AI interaction strategies and validation processes.

Unexpectedly, a subset of students became over-reliant on AI suggestions, producing superficially correct artifacts that concealed underlying misconceptions. This finding highlights the importance of continuous monitoring and targeted intervention to prevent surface-level learning. Furthermore, the small sample size and single-institution context limit broad generalizability; future research should replicate this approach across diverse cohorts and course formats.

In sum, GenAI holds substantial promise for enhancing software engineering education, but only when it is supported by pedagogical frameworks that emphasize critical oversight and conceptual rigor. By aligning learning objectives with a new profile of competencies, namely the Context Provider, Prompt

Design, Critical Validator, and Ethical Guardian, educators can better prepare students for a professional landscape where human–AI collaboration is increasingly the norm.

Conclusion

This exploratory case study provides preliminary evidence that the structured integration of Generative AI (GenAI) may support the development of software engineering competencies among students. The findings suggest that GenAI not only accelerates artifact production but was also associated with more positive learning experiences, including higher self-reported confidence, efficiency, and conceptual understanding. The research identified a fundamental transformation in the engineering workflow, shifting the primary challenge from manual execution to the critical evaluation of AI-generated outputs. This shift underscores the need to redefine SE competencies, emphasizing skills like critical validation, context provision, and prompt engineering.

Limitations and Future Works

Despite these contributions, this study has several limitations that must be acknowledged. First, the case study approach with a very small sample size (four students) and reliance on subjective perceptual data significantly limits the statistical generalizability of the findings to a broader population. Second, the research was conducted in a controlled academic environment, which may not fully reflect the complexities and dynamics of real-world industry contexts. Third, the study did not involve direct validation from industry practitioners; therefore, the definition of "industry needs" was based on theoretical frameworks and literature rather than direct market assessment.

Based on these findings and limitations, we propose several directions for future research. First, there is a clear need to validate and expand upon these initial findings through larger-scale studies that include direct involvement from industry practitioners in the evaluation process. This would test the transferability of the identified human-AI collaboration dynamics to contexts aligned with actual industry standards. Second, the "learning paradox" identified in this study—where effective AI use requires strong foundational knowledge—highlights an urgent need for research focused on developing new pedagogical models. Future studies could investigate how to best leverage GenAI not just as a productivity tool but as an instrument to reinforce fundamental concepts, potentially leading to strategic frameworks for its effective and responsible integration into SE curricula.

A further limitation concerns the scoring of student artifacts. Because all assessments were conducted by a single researcher, no inter-rater reliability measure, such as Cohen's Kappa or an intraclass correlation coefficient, could be calculated. Steps were taken to support consistency, including predefined rubric anchors and documented justifications for each score. However, these measures do not substitute for independent verification by a second qualified rater. Future studies in this area should involve at least two assessors with relevant domain expertise and report agreement statistics to establish the reliability of the scoring process.

References

Afifa, L. (2023). *Layoffs Caused Almost One Million People to Lose Jobs Last Year, Apindo Says - News En.tempo.co*. https://en.tempo.co/read/1675803/layoffs-caused-almost-one-million-people-to-lose-jobs-last-year-apindo-says

Al-Sarayreh, K. T., Meridji, K., & Abran, A. (2021). Software engineering principles: A systematic mapping study and a quantitative literature review. *Engineering Science and Technology, an International Journal*, *24*(3), 768–781. https://doi.org/10.1016/j.jestch.2020.11.005

Assyne, N., Ghanbari, H., & Pulkkinen, M. (2022a). The essential competencies of software professionals: A unified competence framework. *Information and Software Technology*, *151*. https://doi.org/10.1016/j.infsof.2022.107020

Assyne, N., Ghanbari, H., & Pulkkinen, M. (2022b). The state of research on software engineering competencies: A systematic mapping study. *Journal of Systems and Software*, *185*. https://doi.org/10.1016/j.jss.2021.111183

Bauer, F. L. (1972). Software engineering, Information processing. *Proceedings of the Software Engineering*, 1899–2227.

Cantrell, S. G. M. J. R. H. J. (2022). *The skills-based organization: A new operating model for work and the workforce*. https://www.deloitte.com/us/en/insights/topics/talent/organizational-skill-based-hiring.html

Chen, T. (2024). The Impact of AI-Pair Programmers on Code Quality and Developer Satisfaction: Evidence from TiMi studio. *ACM International Conference Proceeding Series*, 201–205. https://doi.org/10.1145/3665348.3665383

Coutinho, M., Marques, L., Santos, A., Dahia, M., França, C., & De Souza Santos, R. (2024). The Role of Generative AI in Software Development Productivity: A Pilot Case Study. *AIware 2024 - Proceedings of the 1st ACM International Conference on AI-Powered Software, Co-Located with: ESEC/FSE 2024*, 131–138. https://doi.org/10.1145/3664646.3664773

Creswell, J. (2014). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*.

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., & Zhang, J. M. (2023). Large Language Models for Software Engineering: Survey and Open Problems. *Proceedings - 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering, ICSE-FoSE 2023*, 31–53. https://doi.org/10.1109/ICSE-FoSE59343.2023.00008

Gao, Y. (2024). *Research: Quantifying GitHub Copilot's impact in the enterprise with Accenture*. https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-in-the-enterprise-with-accenture/

Garousi, V., Giray, G., Tuzun, E., Catal, C., & Felderer, M. (2020). Closing the Gap between Software Engineering Education and Industrial Needs. *IEEE Software*, *37*(2), 68–77. https://doi.org/10.1109/MS.2018.2880823

Hedges, L. V. (1981). Distribution Theory for Glass's Estimator of Effect size and Related Estimators. *Journal of Educational Statistics*, *6*(2), 107–128. https://doi.org/10.3102/10769986006002107

International Organization for Standardization. (2011). *ISO/IEC 25010:2011 – Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*.

Kirova, V. D., Ku, C. S., Laracy, J. R., & Marlowe, T. J. (2024). Software Engineering Education Must Adapt and Evolve for an LLM Environment. *SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, *1*, 666–672. https://doi.org/10.1145/3626252.3630927

Kozov, V., Ivanova, G., & Atanasova, D. (2024). Practical Application of AI and Large Language Models in Software Engineering Education. In *IJACSA) International Journal of Advanced Computer Science and Applications*. *15*(1). https://doi.org/10.14569/IJACSA.2024.0150168

Kuhail, M. A., Mathew, S. S., Khalil, A., Berengueres, J., & Shah, S. J. H. (2024). "Will I be replaced?" Assessing ChatGPT's effect on software development and programmer perceptions of AI tools. *Science of Computer Programming*, *235*. https://doi.org/10.1016/j.scico.2024.103111

Oguz, D., & Oguz, K. (2019). Perspectives on the Gap between the Software Industry and the Software Engineering Education. *IEEE Access*, *7*, 117527–117543. https://doi.org/10.1109/ACCESS.2019.2936660

Ozkaya, I., & Schmidt, D. (2024). *Generative AI and Software Engineering Education*. https://insights.sei.cmu.edu/blog/generative-ai-and-software-engineering-education/

Pereira, J., López, J.-M., Garmendia, X., & Azanza, M. (2024). *Leveraging Open Source LLMs for Software Engineering Education and Training*. 1–10. https://doi.org/10.1109/cseet62301.2024.10663055

Rivera-Ibarra, J. G., Rodríguez-Jacobo, J., Fernández-Zepeda, J. A., & Serrano-Vargas, M. A. (2010). Competency framework for software engineers. *Software Engineering Education Conference, Proceedings*, 33–40. https://doi.org/10.1109/CSEET.2010.21

Sengul, C., Neykova, R., & Destefanis, G. (2024). Software engineering education in the era of conversational AI: current trends and future directions. In *Frontiers in Artificial Intelligence* (Vol. 7). Frontiers Media SA. https://doi.org/10.3389/frai.2024.1436350

Sommerville, Ian. (2011). *Software engineering*. Pearson.

Washizaki, H. (2024). *SWEBOK v4*.

Washizaki, H., Sanchez-Segura, M.-I., Garbajosa, J., Tockey, S., Nidiffer, K. E., & Reilly, A. D. (2025). *An Overview of the SWEBOK Guide - SEBoK*. https://sebokwiki.org/wiki/An_Overview_of_the_SWEBOK_Guide

Zönnchen, B., Thurner, V., & Böttcher, A. (2024). On the Impact of ChatGPT on Teaching and Studying Software Engineering. *IEEE Global Engineering Education Conference, EDUCON*. https://doi.org/10.1109/EDUCON60312.2024.10578680

How to cite:

Afif, G. G., & Putra, P. O. H. (2026). Generative AI in Software Engineering Education: A Study on Competency and Perception. *Jurnal Sistem Informasi (Journal of Information System), 22*(2), 1–17.