Generative AI in Student Software Development Projects: A User Study on Experiences and Self-Assessment

Conference Paper · June 2025

DOI: 10.1145/3723010.3723012

3 authors:

Uwe M. Borghoff
University of the Bundeswehr Munich
353 PUBLICATIONS **2,145** CITATIONS
SEE PROFILE

Mark Minas
University of the Bundeswehr Munich
179 PUBLICATIONS **1,589** CITATIONS
SEE PROFILE

Jannis Schopp
University of the Bundeswehr Munich
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Generative AI in Student Software Development Projects: A User Study on Experiences and Self-Assessment

Uwe M. Borghoff
Department of Computer Science
Institute for Software Technology,
University of the Bundeswehr Munich
Neubiberg, Germany
uwe.borghoff@unibw.de

Mark Minas
Department of Computer Science
Institute for Software Technology,
University of the Bundeswehr Munich
Neubiberg, Germany
mark.minas@unibw.de

Jannis Schopp
Department of Computer Science
Institute for Software Technology,
University of the Bundeswehr Munich
Neubiberg, Germany
jannis.schopp@unibw.de

Abstract

The way software is developed is changing rapidly due to the general availability of generative AI tools. As a result, the software engineering education that is part of every computer science program needs to change. Especially in software engineering courses, such AI tools need to be used and practiced in a meaningful and useful way. The programming project is one such course at our university, and the curriculum will be expanded accordingly in the future. In this paper we describe our approach and a user study among the participants of the last programming project, in which we collected experiences with the use of current AI tools, in particular highlighting their usefulness and limitations. Our study focuses on identifying which aspects of the course students used AI tools for, evaluating successful applications, and uncovering remaining challenges.

CCS Concepts

• **Software and its engineering → Software development process management**; *Software development methods*; • **Applied computing → Education**.

Keywords

software development project course, software engineering education, AI support, AI-based tutoring, experiments

ACM Reference Format:
Uwe M. Borghoff, Mark Minas, and Jannis Schopp. 2025. Generative AI in Student Software Development Projects: A User Study on Experiences and Self-Assessment. In *ECSEE 2025: European Conference on Software Engineering Education (ECSEE 2025), June 02–04, 2025, Seeon, Germany.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3723010.3723012

1 Introduction

The programming project at our university is a software development course in which group work in a software development team is a crucial aspect. Similar courses are probably part of almost all bachelor programs in computer science worldwide. In our programming project, about seven students work together in a team to create a computer game using a structured development process. The main goal is for the participants to gain experience in software development in a team with all the necessary activities, including planning, programming, testing and documentation, as well as the necessary soft skills such as teamwork and communication.

The general availability of generative AI tools (e.g. ChatGPT or GitHub's Copilot) also has an impact on the way software is developed. So far, such AI tools are not part of the curriculum of our programming project. But in order to be able to plan their future use based on experience, we proceeded as follows in the last run of the programming project from October to December 2024. From the beginning, the participants were asked to understand freely available AI tools as useful tools and to use them as such, e.g. when coding or documenting. At the end of the programming project, we conducted a study in the form of a structured questionnaire asking about the experiences with AI tools in the different phases and in relation to the different activities.

This paper describes the study and its results based on the programming project concept. Based on these results, we draw conclusions about how AI tools should be used in the programming project in the future, and in particular, which tools should be created and trained specifically for the programming project.

The rest of the paper is organized as follows: After reviewing related work in Sect. 2, we outline the context of the study, focusing on our programming project, in Sect. 3. Sect. 4 describes the demographics and methodology of the user study, while Sect. 5 summarizes the results. We conclude the discussion in Sect. 6.

2 Related Work

Rane *et al.* [29] examine the general use of AI technologies, such as machine learning algorithms and natural language processing, to develop customized learning experiences. Their study focuses on adaptive learning, highlighting how AI systems can modify instructional approaches in response to real-time feedback and the learner progress. Nitzl *et al.* [24] demonstrate how AI can improve performance in the intelligence *Analysis* phase.

The application of AI—especially conversational/generative AI—in the specific context of a software engineering environment has been the subject of investigation for some time [31, 32]. For a recent literature review, refer to [17], and for a special issue on generative AI in software engineering, see [5]. And there are already promising applications for which AI and related deep learning will become indispensable to software engineering [35]. A study [23] compares the code generated by ChatGPT with that written by developers to

ECSEE 2025, Seeon, Germany
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1282-1/25/06
https://doi.org/10.1145/3723010.3723012

assess which tasks are better performed by engineers or by AI processes, enabling more efficient collaboration, see also [26]. Waseem *et al.* [34] show that ChatGPT significantly addresses skill gaps in software development education, improving efficiency, accuracy, and collaboration. Code quality through AI-powered tools is discussed in [20]. In another study, Bhandari *et al.* [2] discuss new emerging areas of fusion between AI and software engineering. Taken together, these studies suggest that integrating AI into software engineering projects has significant implications for learning and skill development. While AI tools can automate routine tasks and increase productivity, they also require a shift in skill sets. This could lead to new insights into novel AI approaches involving human-in-the-loop to support SE tasks.

The position paper by Daun and Brings [8] discusses ChatGPT's ability to understand and generate human language and thus provide personalized feedback to students. In this way, it can be used as a tutor to support the education of software engineers. In [4], Borghoff, Minas, and Mönch demonstrate the potential of automated tools for program evaluation, grading, and code generation in software development courses. These AI-driven systems offer valuable support by streamlining assessment processes and enhancing the learning experience.

Systems for automatically assessing student programs, known as *autograders* [1], have a long history and can be broadly divided into several categories. In test-based assessment systems, instructors define test cases for assignments, and students' programs are executed against these cases. Programs are considered correct if they pass all tests, with early systems dating back to the 1960s, see [10] for a review.

A contemporary example is ArTEMiS[1] [18], which provides an online code editor with interactive tutorials that provide immediate and consistent feedback in large classes. Current platforms such as Praktomat[2] are web-based, support languages such as Java, and use JUnit for test case implementation. Newer methods also incorporate machine learning techniques, for example, [33].

Chrysafiadi *et al.* [7] introduce a fuzzy-based personalized assessment approach that develops customized test cases for each student based on their knowledge level, programming experience, common error patterns, and item difficulty. This adaptability is a key focus of recent research [14]. Praktomat not only tests student programs against JUnit cases, but also integrates plagiarism detection tools such as JPlag[3] and Checkstyle[4] to ensure coding standards are met.

With deep learning and *Large Language Models* (LLMs) capable of generating code from large code bases, new challenges arise. Tools such as DeepMind's AlphaCode [19], Google's PaLM,[5] Microsoft's CodeBERT [11], and OpenAI's ChatGPT and Codex [12] can generate reasonable code from natural language descriptions.

Gao *et al.* [13] present techniques to improve model performance and usability by comparing *Retrieval-Augmented Generation* (RAG) for LLMs with fine-tuning and prompt engineering. They evaluate these methods based on their ability to incorporate external knowledge and adapt to new contexts. While fine-tuning enables LLMs to deliver consistent and repeatable results, it struggles to incorporate new information. In contrast, RAG can adapt almost instantly to updated contexts. In addition, Ovadia *et al.* [25] demonstrate that RAG outperforms unsupervised fine-tuning in generating results that integrate both existing and new knowledge.

These developments also provide opportunities to use LLMs to improve code quality and support novice learners [16]. AI systems such as Amazon's CodeWhisperer[6] offer real-time suggestions to prevent security vulnerabilities, while tools such as Replit's GhostWriter,[7] Codeium,[8] and Codota/Tabnine[9] assist with real-time code completion, reducing syntax errors. Other notable tools include mutable.ai,[10] CodePal,[11] and Sourcegraph's Cody[12].

More and more programming tasks that traditionally required human expertise can now be performed by AI systems, either autonomously or with human input. Raman and Kumar [28] analyze the *Coding & Testing* phase using a six-step framework. They evaluate the impact of these automated systems at each stage, including GitHub's Copilot [21], and conclude that future education should prioritize code comprehension over mere coding skills [15]. Moroz *et al.* [22] conclude that even when using Copilot, finding a specific solution is not always the main goal, but finding strategies that provide solution instances is equally important.

But critics and bad experiences have also been reported. For example, Choudhuri *et al.* [6] state in their study that there were no statistical differences in the productivity or self-efficacy of the participants when using ChatGPT compared to conventional resources, but instead they found a significantly higher level of frustration. They name pitfalls such as limited advice on niche topics, incomplete assistance, or hallucination, among others. Pudari and Ernst [27] show that while syntax support for the programming language is well on the way to providing useful AI support, the more abstract problems, such as language idioms and complex design rules, are still far from being solved by AI.

3 The Programming Project

For most of our students, the programming project is their first software development project, where they work in a team of about seven students and realize a complete computer game in Java. The syllabus for this course is worth 9 credits, which corresponds to a workload of about 270 hours in the period from October to December.

We have offered and improved this course over many years (see Sect. 3.3); the following paragraphs describe its organization in the last term, from October to December 2024.

3.1 Basic Organization

Each team was assigned a tutor and a supervisor. The tutors were students who had taken the course the previous year, and they

[1]https://github.com/ls1intum/ArTEMiS
[2]https://www.waxmann.com/automatisiertebewertung/
[3]https://github.com/jplag/JPlag
[4]https://checkstyle.org/
[5]https://ai.google/discover/palm2/

[6]https://aws.amazon.com/de/codewhisperer/
[7]https://blog.replit.com/ai
[8]https://codeium.com/
[9]https://www.tabnine.com/blog/codota-is-now-tabnine/
[10]https://mutable.ai/
[11]https://codepal.ai/
[12]https://about.sourcegraph.com/cody

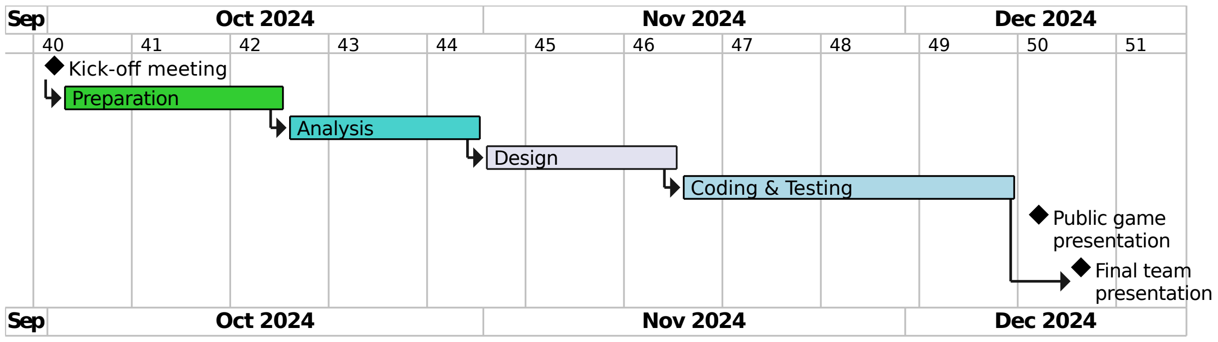

Figure 1: Schedule of the programming project.

assisted the team members with the usual problems encountered in a software development project. The supervisors were members of the academic staff who provided higher-level support to the teams, monitored and evaluated the teams' progress, approved artifacts, and controlled when the teams were allowed to proceed to the next phase and when the project was successfully completed.

Prior to the course, a list of twelve game proposals was compiled, each of which was sketched on a page by the tutors. All games had to be multiplayer 3D games with network support, based on popular board and card games (e.g., Monopoly, Ludo, Risk), and about equally difficult to implement. Students enrolled in the course were asked to form seven teams and prioritize a list of three games from this list of all games before the course began. Based on these priorities, games were assigned to these teams so that each team realized a different game.

The course then began with the *kick-off meeting*, where all students, supervisors, and tutors met in the lecture hall to discuss organizational issues. In the following weeks, each team worked independently on their projects, following the traditional waterfall process model as described below (Sect. 3.2). This process model was deliberately chosen for three reasons. First, it is simple and clearly structured, and it is the first software development project in a team for most of our students. As a result, they are unfamiliar with the different phases and artifacts involved in software development. Second, the course lasts only 11 weeks, which is very little time to learn and try out a more modern, agile process model. Finally, and most importantly, the more rigid waterfall setup makes it easier to measure the students' use of AI in concrete parts of the project.

At the end of the term, the teams presented their games to the other teams and invited guests, e.g. other students, in a so-called *public game presentation*. This event also included a competition where each participant (including all guests) had the opportunity to rate each game presented, and the winning team was awarded a prize.

The project schedule for each team was organized into four sequential phases as shown in Figure 1 and explained below. Each team had a fixed weekly one-hour meeting with the supervisor and tutor, where team members presented their progress and discussed problems and solutions with the supervisor and tutor. Each phase ended at a milestone, and teams were not allowed to proceed to the next phase until they had successfully completed the previous phase. It was the supervisor's responsibility to evaluate and approve the team's progress and artifacts. As a result, each team's individual phases could take longer than planned, causing teams to fall behind schedule. This was the case for some teams (especially in the *Design* phase), but all were able to catch up by the end of the term and participate in the *public game presentation* with an executable game.

3.2 Project Phases

The phases, as shown in Figure 1, were as follows. The first phase was the *Preparation* phase (2 weeks), during which each team member worked on training tasks and practiced programming techniques. This phase was independent of the specific games that the teams would implement in the later phases. Instead, a realization of the simple *Battleship* game was available to all teams from the start. Like the games to be realized later, it is a multiplayer 3d game, playable over the network, and implemented on top of the freely available 3d game engine jMonkeyEngine. *Battleship* was designed to demonstrate the architecture and various concepts of 3d games, and also served as a starting point and blueprint for the teams' own game development. The task of the *Preparation* phase was therefore to familiarize the students with *Battleship* and its structure and to extend it.

The following three phases follow the waterfall process model. During the *Analysis* phase (2 weeks), the teams created a product requirement specification based on the one-page sketch of their game. The teams had to produce artifacts such as an extended game description as prose text, a data model described by a class diagram, a list of all use cases and their descriptions based on a use case template, a description of acceptance tests as a table of test cases, and a first version of the user manual including sketches of the GUI.

The software specification document was created by the teams in the following *Design* phase (2 weeks) and consisted of a prose description of the game specification, in particular the chosen architecture and the communication protocol of the required network protocol. Recommended diagrams for specifying details were e.g. class diagrams, BPMN diagrams, sequence diagrams and state diagrams. System, integration and unit tests were specified in a test case table. In the *Coding & Testing* phase (about 3 weeks), the games were finally implemented and tested according to the teams' software specification documents. Any deviations from these documents were required to be documented, and the final version of the user manual had to be made available.

Table 1: Demographics of course and study participants.

Characteristic	Course participants (n=49)	Study participants (n=38)
Gender		
Male	46	35
Female	3	3
Academic programs		
Computer Science	34	26
Business Informatics	13	10
Mathematical Engineering	2	2
Participation rate	-	≈78%

A team is considered to have successfully completed their programming course when their supervisor has approved their final product and they have presented their project and game to the supervisor in a *final team presentation.* Upon meeting these criteria, all team members were awarded the course credits.

3.3 Course Evolution

The course of our programming project has evolved over the years, but we have kept its basic organization with teams realizing games, using Java and the waterfall process model, and presenting all games in the *public game presentation.* The latter has proven to be an essential aspect of the programming project for the following reason: Participants do not receive a grade at the end of the course because it is difficult to evaluate the individual performance of the participants. Therefore, only successful participation is confirmed. Without the "reward" of a better grade, there is less motivation to commit to the project beyond the minimum effort. The public presentation of one's own game and, above all, the competition between the teams for the prize of the best, most beautiful and funniest game counteracts this. This is evidenced by the fact that all teams were able to finish their game by the public presentation date, even if they were behind schedule before because one of the phases took longer than planned. In addition, many students stated that the public presentation of their game was a highlight of the course and was fun.

All of the major evolutionary steps over the years have been in response to external events.

The first major evolutionary step was triggered by the Covid pandemic. The entire course and teamwork had to be done online. We had to respond to user feedback as described in [3]: Unlike before, all teams had to make the same game (a dungeon crawler in 2022 and a racing game in 2023), but with different themes. To prevent the teams from sharing their solutions, all participants had to complete assignments that were evaluated by an automatic program assessment system.

With the end of the pandemic, we abandoned the assignments and had different games realized. This decision was also influenced by the fact that the teams spent a lot of time solving the assignments, which meant that the actual project work and the quality of the games suffered.

In the evaluation of the last term, a large number of students confirmed that they preferred to return to the programming project without assignments and that it was more fun that way. In fact, the quality of the 2024 games has increased significantly compared to the two years with assignments, considering the number of implemented functions and the graphical design of the games.

There was also a growing trend of students adopting generative AI tools in their own workflows. While not formally integrated into the course curriculum, the use of such tools was encouraged during this year's course for specific tasks such as code documentation and text proofreading.

4 Study

A user study was conducted during the final presentations to investigate the extent and nature of AI tool usage. Understanding how students adopt and apply general-purpose AI tools, mostly large language models, during the programming project can provide valuable insights for developing dedicated AI tools to better support our software engineering course. However, these tools are neither formally integrated into the course curriculum nor openly discussed by students, highlighting the need for further investigation to assess their usage and impact.

Rather than advocating a specific approach to AI integration we explore how students naturally incorporate AI tools into a software development project. We aim to establish a baseline understanding of AI adoption in order to determine what types of support, tutoring or other AI-driven interventions may be needed in the future.

A recent experiment by Rapaka *et al.* [30] examines two versions of an educational game. One version followed a traditional design, while the other incorporated an intelligent, AI-based process. The experiment demonstrates that immersive and AI technologies can serve as valuable tools in the development of educational games and entertainment applications. At the same time, previous research [6] highlights that using AI in learning software engineering can significantly increase frustration levels.

To address these questions, we conducted a user study during the October-December term of 2024 to capture a snapshot of AI adoption among the student body. The goal of the study was to determine which aspects of the course students engaged with using AI tools, identify successful applications, and uncover any remaining challenges.

4.1 Demographics

The study included 38 participants, representing approximately 78% of the 49 students enrolled in the course. As shown in Table 1, most of the participants were Computer Science students, followed by

Business Informatics and Mathematical Engineering students. The gender and academic program distribution of the study participants closely mirrored that of the overall course population.

Of the 49 students, 43 were in the middle of their bachelor's program, while 6 were nearing its completion. At the time of the study, all participants had completed the required undergraduate courses in computer science and object-oriented programming.

4.2 Methodology

Data for the study were collected using a structured questionnaire distributed during the final team presentation of the programming project. This timing allowed participants to reflect on their experiences during all phases of the project.

Participation in the study was voluntary, and students were informed of their right to withdraw at any time. Anonymity of data was ensured, and participants were given the opportunity to request that their data be deleted. The study adhered to general data protection regulations to ensure the ethical handling and protection of personal data.

The questionnaire was structured to comprehensively evaluate the adoption and application of AI tools during the programming project, following the *Technology Acceptance Model* [9]. This framework emphasizes factors that influence technology acceptance, such as perceived usefulness and ease of use.

4.3 Questionnaire

We invited all attending students to participate in the study, regardless of whether or not they used AI tools. Including students who did not use AI was essential, as their insights provide valuable context for building an authentic picture of AI adoption. For those who did not use AI, we asked them to share their reasons for not using AI, providing perspective on barriers to adoption or other influencing factors.

(1) **General question**
Did you use AI tools during your programming project?
Options: Yes, No. If "No", participants provided reasons such as lack of knowledge about available tools, no access to suitable tools, privacy or trust concerns, and other (open-ended response).

Perceived usefulness was incorporated into the study by asking students to rate how AI tools supported their tasks throughout the programming project phases, which included *Preparation*, *Analysis*, *Design*, and *Coding & Testing*. The questionnaire captured several dimensions that are helpful in understanding the adoption and use of generative AI tools. Participants were asked to describe the tasks they completed using AI tools and to identify the platforms they used. Questions focused on the effectiveness of AI in Debugging, Documentation, and Code Generation, gauging how well these tools contributed to student success and efficiency in specific project contexts.

(2) **Repeated questions for each project phase**
- (a) **Did you use AI tools during this phase?**
Options: Yes, No.
- (b) **What specific AI platforms did you use during this phase?**
Examples: ChatGPT, Copilot, Claude, Gemini, LLaMA, or other (open-ended response).
- (c) **What tasks have you worked on using AI tools?**
Participants could choose from predefined tasks or provide their own description in an open-ended response.
- (d) **How helpful was AI in this phase?**
Scale: 1 (not helpful) to 5 (very helpful).
- (e) **How much time did the AI save you during this phase?**
Options: None, Less than 1 hour, 1—3 hours, 4—6 hours, 7—9 hours, 10+ hours.
- (f) **Did you face challenges when using AI tools?**
Options: Yes, No. If "Yes", participants could specify challenges such as unclear results, difficulty integrating tools, and other (open-ended response).

Responses on the helpfulness were recorded using a five-point Likert scale ranging from "not helpful" to "very helpful". Ease of use was addressed by exploring the challenges of integrating AI tools into the workflow. Open-ended questions invited participants to describe barriers to usability, such as unclear results. This provided qualitative insights into the effort required to effectively integrate the tools into students' workflows.

To assess the overall usefulness of these tools, participants provided a holistic rating of their experience throughout the project. The survey also explored participants' behavioral intentions, including their openness to using AI tools in future projects.

(3) **Behavioral intentions and final reflections**
- (a) **How do you rate the overall usefulness of AI tools in the programming project?**
Scale: 1 (not helpful) to 5 (very helpful).
- (b) **Would you use AI tools in future programming projects?**
Scale: 1 (very unlikely) to 5 (very likely).
- (c) **Could you have achieved similar results without AI?**
Options: Yes (without extra effort), Yes (with more effort), No (AI was essential).
- (d) **What new or additional AI tools or features would you find helpful?**
Open-ended response for suggestions and improvements.
- (e) **Do you have any additional comments or feedback regarding the use of AI in your project?**
Open-ended response for qualitative feedback.

5 Results

Of the 38 participants in this study, 34 used AI tools during the programming project, resulting in an adoption rate of 89.4%. The remaining four participants all reported that they did not use AI tools, citing either a lack of need or personal reservations. In addition, one participant stated that they were unaware of the AI tools available.

As shown in Figure 2, the use of AI tools varied significantly across the four phases of the programming project. During the *Preparation* phase, 22 participants used AI tools, while 16 did not. In the *Analysis* phase, the use of AI dropped slightly, with 13 participants using the tools and 25 not using them. The *Design* phase saw the lowest adoption rate, with only 10 participants using AI tools,

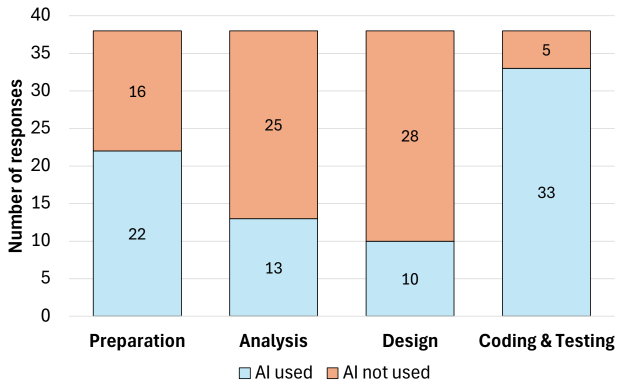

Figure 2: AI tool adoption across the project phases.

compared to 28 who abstained. Conversely, the *Coding & Testing* phase had the highest adoption rate, with 33 participants using AI tools and only 5 opting out. Interestingly, ChatGPT was used by all students who used AI tools, regardless of the phase. The next most commonly used tool was GitHub's Copilot, especially in the coding phase. Other tools were rarely used. Below we take a closer look at the role of AI in each phase.

Figure 3 illustrates the perceived helpfulness of AI tools across the four project phases using a Likert scale visualization. During both the *Preparation* and *Coding & Testing* phases, a significant proportion of participants rated AI tools as "helpful" or "very helpful". In contrast, the *Design* phase saw not only the lowest adoption rates, but also the most critical ratings, with "neutral" and "not helpful" responses dominating. This is likely due to the limitations of most AI tools in generating syntactically correct diagrams, such as class or sequence diagrams, which are heavily used in this phase.

Overall, the students showed a willingness to incorporate AI into their workflows and actively attempted to complete the project tasks using AI tools. However, the nature of the tasks and the results varied depending on the phase of the project.

5.1 Preparation Phase

During the *Preparation* phase, 19 responses were collected about the tasks that participants completed using AI tools. Of these, 18 responses were categorized into specific task groups, while one response was too vague to be included in the analysis. The categorized responses highlighted different ways in which students used AI tools in their work, demonstrating both the diversity of their applications and the challenges they faced.

The most commonly reported task (n=9) was comprehension. Students relied on AI tools to understand an existing code base that formed the basis of their later project. This included working with the jMonkeyEngine, design patterns, and architectural choices, which required the use of AI to explain code snippets and clarify technical concepts. However, comprehension tasks were also frequently associated with challenges. Of the nine participants who used AI for comprehension, six reported encountering "inaccurate results" and two mentioned "missing context". These problems occurred when the AI did not relate to the existing code base or was not trained on domain-specific frameworks such as jMonkeyEngine.

The code generation tasks (n=4) were closely related, as students were tasked with implementing additional features into the existing project. Of these participants, three noted issues with "inaccurate results" that required significant corrections to the AI-suggested code snippets. This issue highlights a broader limitation of AI tools in producing reliable code when the underlying logic or requirements are highly specific.

Similarly, code improvement (n=3) emerged as a recurring theme, where students used AI to incorporate feedback from their supervisors into their solutions. However, two participants reported frustration with generic or irrelevant suggestions that could not be integrated into the existing code.

The use of AI for documentation (n=3) was also notable, consistent with staff recommendations to use AI to create and refine code documentation. Although documentation tasks were relatively straightforward, one participant reported challenges with "inaccurate results". Debugging (n=2) was surprisingly underrepresented, despite being an introductory task for this phase, where students were asked to identify bugs in the provided code base. Both participants who used AI for debugging noted difficulties stemming from the AI's inability to understand the larger project scope, resulting in ineffective or misleading suggestions. Finally, one participant mentioned image generation (n=1), although it remains unclear what specific artifact was produced.

These findings highlight the multiple roles that AI tools played during the *Preparation* phase, but also reveal significant limitations in their contextual understanding and output reliability when working with unfamiliar code. The recurring issues of inaccuracy, lack of context, and overly generic suggestions show that while AI tools can improve productivity, they require careful oversight and integration into specific workflows.

5.2 Analysis Phase

During the *Analysis* phase, 13 participants reported using AI tools, with 11 providing specific tasks that were categorized into five different groups. None of the responses had to be discarded due to vague or uninterpretable responses.

Requirements specification (n=8) was the most frequently reported task. In this task, students created their own version of the requirements document, outlining how the game to be developed should work. We believe that AI helped with these tasks because text generation is one of its strengths, and many AI systems are familiar with popular games that students adapted into software during the project. Since no complaints were reported, this seems to have been a successful use of AI tools to help formulate precise and comprehensive requirements. This was followed by use case generation (n=4), where AI tools helped create structured lists that outlined the use case's ID, priority, and key details such as purpose, actors, and conditions. This repetitive and systematic task benefited from AI by streamlining the process. However, one complaint was that the results were unreliable and required manual corrections.

Documentation tasks (n=4) focused primarily on creating user manuals. These required, among other things, detailed descriptions of the game's components, controls, and gameplay elements. As this

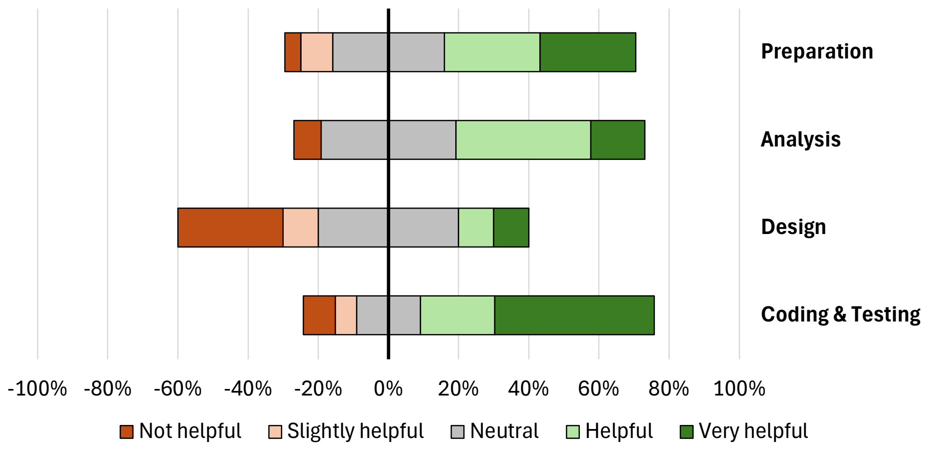

Figure 3: AI tool helpfulness across the project phases.

is also a text-heavy task, it benefited from the same strengths of the AI as those observed in the requirements specification mentioned above. There were no complaints.

Diagram generation (n=2) in this phase attempts to generate class diagrams that are simple in design and complexity, as they primarily outline the preliminary classes and structure of the project. However, both mentions were accompanied by complaints that the diagrams were not correct or detailed enough. The AI's difficulty in generating meaningful diagrams was quickly apparent to the students. Finally, GUI mockups (n=2) were used to design and present the user interface at this early stage, collaborating with the "customer" on how the program would look. The strength of AI in generating images definitely played a role here, allowing students to visualize and iterate on their designs. Despite the low number of mentions, several groups used AI-generated images in their projects, which was evident in the final presentations.

While only about one-third of participants reported using AI in this phase, the overall helpfulness was rated positively and few problems were reported. Text-heavy tasks such as requirements specification and documentation clearly benefited from the strengths of AI tools. Challenges such as lack of detail and unreliable output underscore the importance of critically evaluating and refining AI-generated output to align with project requirements.

5.3 Design Phase

Of the 10 participants who reported using AI tools, nine provided specific tasks that were categorized into four groups. Again, no response had to be discarded due to an uninterpretable response.

The most commonly reported task was creating diagrams and charts (n=5), which in this phase included flowcharts, class, package, sequence, and state diagrams. These artifacts are meant to visualize the structural and behavioral aspect of the software, and serve as a transition from conceptual work to concrete implementation. However, and as reported in the previous phase, this may be related to the fact that two participants reported insufficient diagrams and one student specifically mentioned the lack of content in the generated diagram.

Evaluating and comparing design approaches (n=4) was another key task category, which involved evaluating different design strategies to select effective solutions, such as thick client and thin client models in a network-based architecture. One participant reported the challenge of designing a prompt with the right amount of context to elicit a sufficient response.

Optimizing design prototypes (n=3) was also mentioned, where students asked AI tools to improve their ideas. Participants did not report any problems with this task.

Finally, creating images (n=1) was mentioned because one student reportedly continued to improve GUI elements. No problems were reported with this task.

While most of the reported tasks were completed without reported problems, the results must be considered in light of the relatively low adoption of AI tools. Only 10 of the 38 students who participated in the study reported using them. Furthermore, as shown in Figure 3, participants rated the helpfulness of AI at this stage the lowest, which may be due to the high demand for diagram-based artifacts.

5.4 Coding & Testing Phase

This phase saw the highest adoption of AI tools, with 33 of 38 participants reporting their use. All 33 participants identified at least one task where AI provided assistance. These were grouped into 6 tasks. However, problems were prevalent, with 27 participants reporting difficulties and 24 citing one or more challenges in their feedback.

The tasks of Code Generation, Code Improvement, and Code Understanding, which were grouped together due to common themes and frequent concurrent mentions (n=28), were widely used by participants. Because this phase was primarily focused on implementing the designs from the previous phase, students reported a variety of challenges. The most commonly reported challenge, cited by 10 participants, was incorrect code. The generated code often did not integrate with the existing code base, either because of logical errors or because existing classes and methods were overlooked or misinterpreted.

One participant specifically mentioned encountering "hallucinations" where the AI generated incorrect code that relied on framework methods that did not exist. Six participants noted difficulties because some AI models were not trained on domain-specific frameworks or libraries, such as jMonkeyEngine or Lemur, resulting in output that was incompatible with their projects. Three participants reported that AI tools struggled to handle the complexity of their projects because they could not provide the entire code base or large numbers of classes to the AI tool.

This resulted in output that conflicted with existing code and did not integrate seamlessly. Finally, two participants highlighted problems with prompting, attributing their struggles to an inability to effectively communicate their requirements to the AI. Similar challenges were observed across tasks, particularly in code understanding and improvement efforts, as the AI performed better with standard Java but often failed with specialized frameworks.

Documentation tasks (n=24) were also among the most frequently reported, with participants using AI tools to create or refine their project documentation. At this stage, documentation primarily involved the creation of student-written Javadocs for the classes. Encouraged by the staff, many students found the AI to be particularly effective in analyzing the methods and generating comprehensive and contextual documentation. Only one participant reported encountering an erroneous AI-generated document, indicating that the tools performed well in this context.

Debugging tasks (n=20) were closely related to code generation, as both required AI tools to work within existing code bases and project frameworks. Five participants reported that the AI struggled with unfamiliar frameworks or libraries, such as jMonkeyEngine, echoing similar issues observed with code generation. Three participants reported incorrect output during debugging, including one instance where the AI "hallucinated" by generating nonsensical code that was incompatible with the context. Other cases involved solutions that did not address the actual problem or conflicted with the existing implementation, complicating integration efforts. One participant also noted difficulties with prompting, particularly in framing complex code problems or generating contextually relevant methods.

Test generation (n=7) was another area where AI tools were used, specifically to test the model created within the *Model-View-Controller* architecture. While generally effective, some problems were reported. One participant noted that poorly defined tasks for the AI resulted in outputs that did not meet the intended goals. Another participant observed that in some cases the AI produced nonsensical tests. Finally, general questions (n=1) and music generation (n=1) were reported, but no problems with these tasks were documented.

While the *Coding & Testing* phase saw the highest adoption of AI tools among participants, it also revealed significant challenges, particularly with tasks such as code generation, debugging, and improving existing implementations. The tools proved valuable for automating repetitive tasks and generating initial designs, but recurring problems such as incorrect output, difficulties with domain-specific frameworks such as jMonkeyEngine, and limitations in handling complex code bases underscored the need for careful oversight. Despite these challenges, tasks such as documentation and test generation demonstrated the potential of AI tools to improve productivity when clear prompts and well-defined workflows were used. These findings underscore that while AI tools can significantly assist the *Coding & Testing* phase, their effective use requires both user expertise and improvements in tool design, particularly for specialized contexts.

5.5 Final Observations

In addition to analyzing specific project phases, participants were asked if they believed they could have achieved the same results without the use of AI tools. Seven participants indicated that they could have achieved similar results without AI, while 24 indicated that they could have done so, but with significantly more effort. Notably, three participants reported that AI was critical to their success. These responses underscore the significant role AI played in assisting participants during the project. While some were confident that they could achieve similar results without AI, the majority acknowledged the significant time and effort saved by integrating these tools into their workflows, as shown in Figure 4. A smaller but notable group felt that AI was essential.

Interestingly, as shown in Figure 5, the overwhelmingly positive perception of the role of AI contrasts with the problems reported throughout the study. Self-reported time savings varied by project phase, with the highest savings in the *Coding & Testing* phase and the lowest in the *Design* phase. However, participants frequently encountered challenges such as inaccurate or incomplete output, difficulties integrating AI-generated code with domain-specific frameworks such as jMonkeyEngine, and the inability of AI tools to handle complex project contexts.

Despite these difficulties, respondents were generally positive about AI. The majority recognized the potential for AI to improve productivity, particularly for repetitive or structured tasks such as documentation and test generation. The high likelihood that respondents will use AI tools again in future projects underscores their willingness to integrate AI into their workflows despite its limitations.

6 Conclusions

In this paper we present a user study and its results. The background of the user study was the programming project, a software development project aimed at integrating AI tools into the curriculum. The participants of the last run of the programming project were asked to use freely available AI tools (such as ChatGPT and GitHub's Copilot) in their project. In the user study at the end of the project, they were asked about their experiences using a structured questionnaire.

The results of the study were ambivalent: the vast majority of participants found the AI tools in the project useful and wanted to continue using them. However, they were considered useful only for text-intensive tasks, such as documentation and coding, but with significant drawbacks when coding, as the code generated by the AI tools often could not be integrated into the existing code. Experiences with AI tools were mostly negative when it came to diagrams, such as class or sequence diagrams. Despite these limitations, we observed a strong willingness to adopt AI, suggesting that students will continue to rely on such tools regardless of potential risks and frustrations. This raises concerns about AI dependency,

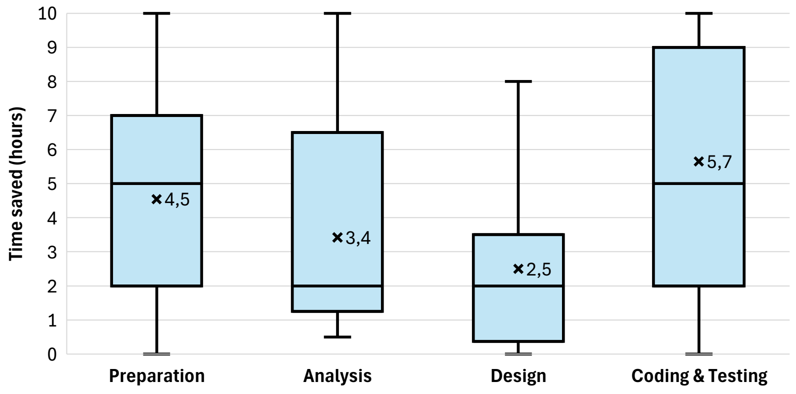

Figure 4: Self-reported time savings by project phase.

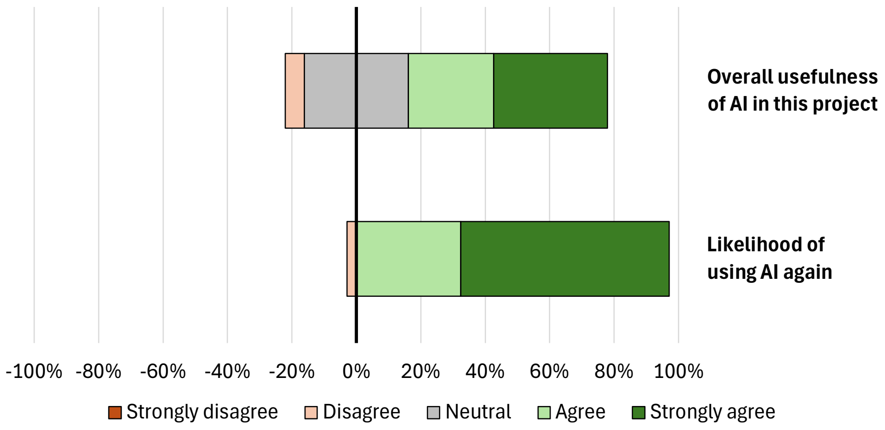

Figure 5: Perceived usefulness of AI in the project and likelihood of future use.

such as skill degradation or inhibition of basic skill development, and highlights the need for careful integration and monitoring.

The project has maintained a similar workload and structure across iterations, and previous classes also saw little or no use of AI tools. This suggests that AI adoption was driven by its increased availability and encouragement in the course rather than by project demands. It may be worth considering that the adoption rate of AI tools in this study was influenced by the general encouragement of AI use early in the course. However, since the focus of the study is not on adoption rates, but rather on how students integrate AI tools into the programming project, our main takeaway is to understand the patterns of AI use.

We have drawn the following lessons from these results. As we have learned from previous years, students use AI tools even in a project course like our programming project. Therefore, we need to include such tools in the curriculum of the programming project and adapt the training accordingly. In order to avoid that solutions generated by AI tools are adopted without reflection and to increase learning success, we plan to pursue the approach of AI-based tutoring in the coming months. Instead of providing direct and sometimes incorrect answers, AI tools should support and motivate students through structured thinking and reflection. Since freely available AI tools have shown weaknesses in code generation, at least in terms of integration into existing code, we will continue to pursue the approach based on RAG [13, 25]. Our goal is to make a context-aware AI tutor a more attractive alternative to the AI tools used by the students in the study. For the time being, we will focus on more text-heavy aspects, as diagrams are not yet well supported. This is particularly unfortunate because diagrams are often used in planning activities in software development. By designing AI's role as a tutor, we aim to more effectively develop students' skills and competencies, while preventing them from relying on AI tools for definitive answers. Our findings reflect the experiences of the students who participated in this specific programming project. Future studies should look at a wider range of software engineering projects and courses to improve generalizability.

Acknowledgments

We would like to thank our best students for their entertaining and humorous games, which we enjoy testing at the Institute's traditional Christmas party.

The authors wish to acknowledge the use of DeepL Translator, DeepL Write and Grammarly in the writing of this paper. These tools were used to improve the language in the paper and to translate some parts of the paper from the original German language. Of course, the paper remains an accurate representation of the authors' underlying work and novel intellectual contributions.

References

[1] Kirsti Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Comput. Sci. Educ.* 15, 2 (2005), 83–102. doi:10.1080/08993400500150747

[2] Kirti Bhandari, Kuldeep Kumar, and Amrit Lal Sangal. 2023. Artificial Intelligence in Software Engineering: Perspectives and Challenges. In *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)*. 133–137. doi:10.1109/ICSCCC58608.2023.10176436

[3] Uwe M. Borghoff, Mark Minas, and Kim Mönch. 2023. Using Automatic Program Assessment in a Software Development Project Course. In *Proceedings of the 5th European Conference on Software Engineering Education, ECSEE 2023, Seeon/Bavaria, Germany, June 19-21, 2023*, Jürgen Mottok (Ed.). ACM, 22–30. doi:10.1145/3593663.3593669

[4] Uwe M. Borghoff, Mark Minas, and Kim Mönch. 2024. Automatic Program Assessment, Grading and Code Generation: Possible AI-Support in a Software Development Course. In *Artificial Intelligence and Soft Computing - 23rd International Conference, ICAISC 2024, Zakopane, Poland, June 16-20, 2024, Proceedings, Part III (Lecture Notes in Artificial Intelligence 15166)*, L. Rutkowski et al. (Eds.). Springer, 39–51. doi:10.1007/978-3-031-81596-6_4

[5] Anita D. Carleton, Davide Falessi, Hongyu Zhang, and Xin Xia. 2024. Generative AI: Redefining the Future of Software Engineering. *IEEE Softw.* 41, 6 (2024), 34–37. doi:10.1109/MS.2024.3441889

[6] Rudrajit Choudhuri, Dylan Liu, Igor Steinmacher, Marco Gerosa, and Anita Sarma. 2024. How Far Are We? The Triumphs and Trials of Generative AI in Learning Software Engineering. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal). 184:1–184:13. doi:10.1145/3597503.3639201

[7] Konstantina Chrysafiadi, Maria Virvou, and George A. Tsihrintzis. 2022. A fuzzy-based mechanism for automatic personalized assessment in an e-learning system for computer programming. *Intell. Decis. Technol.* 16, 4 (2022), 699–714. doi:10.3233/IDT-220227

[8] Marian Daun and Jennifer Brings. 2023. How ChatGPT Will Change Software Engineering Education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2023, Turku, Finland, July 7-12, 2023*, Mikko-Jussi Laakso, Mattia Monga, Simon, and Judithe Sheard (Eds.). ACM, 110–116. doi:10.1145/3587102.3588815

[9] Fred Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (1989), 319–340. doi:10.2307/249008

[10] Christopher Douce, David Livingstone, and James Orwell. 2005. Automatic test-based assessment of programming: A review. *ACM J. Educ. Resour. Comput.* 5, 3 (2005), 4 pages. doi:10.1145/1163405.1163409

[11] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 1536–1547. doi:10.18653/v1/2020.findings-emnlp.139

[12] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *ACE '22: Australasian Computing Education Conference, Virtual Event, Australia, February 14 - 18, 2022*, Judy Sheard and Paul Denny (Eds.). ACM, 10–19. doi:10.1145/3511861.3511863

[13] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv* abs-2312.10997 (2024). doi:10.48550/arXiv.2312.10997

[14] Carlos G. Hidalgo-Suarez, Víctor A. Bucheli, and Hugo Ordoñez. 2022. Automatic Assessment of Learning Outcomes as a New Paradigm in Teaching a Programming Course: Engineering in Society 5.0. *Rev. Iberoam. de Tecnol. del Aprendiz.* 17, 4 (2022), 379–385. doi:10.1109/RITA.2022.3217193

[15] Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin I. Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen T. Kraemer, Violetta Lonati, Claudio Mirolo, and Renske Weeda. 2019. Fostering Program Comprehension in Novice Programmers - Learning Activities and Learning Trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education, ITiCSE-WGR 2019, Aberdeen, Scotland Uk, July 15-17, 2019*, Bruce Scharlau, Roger McDermott, Arnold Pears, and Mihaela Sabin (Eds.). ACM, 27–52. doi:10.1145/3344429.3372501

[16] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, CHI 2023, Hamburg, Germany, April 23-28, 2023*, Albrecht Schmidt et al. (Eds.). ACM, 455:1–455:23. doi:10.1145/3544548.3580919

[17] Peter Kokol. 2024. The Use of AI in Software Engineering: A Synthetic Knowledge Synthesis of the Recent Research Literature. *Inf.* 15, 6 (2024), 354. doi:10.3390/INFO15060354

[18] Stephan Krusche and Andreas Seitz. 2018. ArTEMiS: An Automatic Assessment Management System for Interactive Learning. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education, SIGCSE 2018, Baltimore, MD, USA, February 21-24, 2018*, Tiffany Barnes, Daniel D. Garcia, Elizabeth K. Hawthorne, and Manuel A. Pérez-Quiñones (Eds.). ACM, 284–289. doi:10.1145/3159450.3159602

[19] Yujia Li et al. 2022. Competition-level code generation with AlphaCode. *Science* 378, 6624 (2022), 1092–1097. doi:10.1126/science.abq1158

[20] Boris Martinovic and Robert Rozic. 2025. Perceived Impact of AI-Based Tooling on Software Development Code Quality. *SN Comput. Sci.* 6, 1 (2025), 63. doi:10.1007/S42979-024-03608-4

[21] Antonio Mastropaolo et al. 2023. On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot. In *45th International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023.* IEEE/ACM, 2149–2160. doi:10.1109/ICSE48619.2023.00181

[22] Ekaterina A. Moroz, Vladimir O. Grizkevich, and Igor M. Novozhilov. 2022. The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement. In *Proc. 2022 Conf. of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*. IEEE, 386–390.

[23] Nathalia Nascimento, Paulo Alencar, and Donald Cowan. 2023. Comparing Software Developers with ChatGPT: An Empirical Investigation. *arXiv* abs-2305.11837 (2023). doi:10.48550/arXiv.2305.11837

[24] Christian Nitzl, Achim Cyran, Sascha Krstanovic, and Uwe M. Borghoff. 2024. The Use of Artificial Intelligence in Military Intelligence: An Experimental Investigation of Added Value in the Analysis Process. *arXiv* abs-2412.03610 (2024), 1–28. doi:10.48550/arXiv.2412.03610

[25] Oded Ovadia, Menachem Brief, Moshik Mishaeli, and Oren Elisha. 2024. Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs. *arXiv* abs-2312.05934 (2024). doi:10.48550/arXiv.2312.05934

[26] Alfonso Piscitelli, Gennaro Costagliola, Mattia De Rosa, and Vittorio Fuccella. 2024. Influence of Large Language Models on Programming Assignments - A user study. In *Proceedings of the 16th International Conference on Education Technology and Computers, ICETC 2024, Porto, Portugal, September 18-21, 2024.* ACM, 33–38. doi:10.1145/3702163.3702168

[27] Rohith Pudari and Neil A. Ernst. 2023. From Copilot to Pilot: Towards AI Supported Software Development. *arXiv* abs-2303.04142 (2023). doi:10.48550/arXiv.2303.04142

[28] Arun Raman and Viraj Kumar. 2022. Programming Pedagogy and Assessment in the Era of AI/ML: A Position Paper. In *COMPUTE 2022, Jaipur, India, November 9-11, 2022*, Venkatesh Choppella, Amey Karkare, Chitra Babu, and Sridhar Chimalakonda (Eds.). ACM, 29–34. doi:10.1145/3561833.3561843

[29] Nitin Rane, Saurabh Choudhary, and Jayesh Rane. 2023. Education 4.0 and 5.0: Integrating artificial intelligence (AI) for personalized and adaptive learning. *Available at SSRN 4638365* (2023). doi:10.2139/ssrn.4638365

[30] Anuj Rapaka, S. C. Dharmadhikari, Kishori Kasat, Chinnem Rama Mohan, Kuldeep Chouhan, and Manu Gupta. 2025. Revolutionizing learning - A journey into educational games with immersive and AI technologies. *Entertain. Comput.* 52 (2025), 100809. doi:10.1016/J.ENTCOM.2024.100809

[31] Daniel Russo. 2024. Navigating the Complexity of Generative AI Adoption in Software Engineering. *ACM Trans. Softw. Eng. Methodol.* 33, 5 (2024), 135:1–135:50. doi:10.1145/3652154

[32] Cigdem Sengul, Rumyana Neykova, and Giuseppe Destefanis. 2024. Software engineering education in the era of conversational AI: current trends and future directions. *Frontiers Artif. Intell.* 7 (2024). doi:10.3389/FRAI.2024.1436350

[33] Shashank Srikant and Varun Aggarwal. 2013. Automatic Grading of Computer Programs: A Machine Learning Approach. In *2013 12th International Conference on Machine Learning and Applications, Miami, FL, USA*, Vol. 1. 85–92. doi:10.1109/ICMLA.2013.22

[34] Muhammad Waseem, Teerath Das, Aakash Ahmad, Peng Liang, Mahdi Fahmideh, and Tommi Mikkonen. 2024. ChatGPT as a Software Development Bot: A Project-Based Study. In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2024, Angers, France, April 28-29, 2024*, Hermann Kaindl, Mike Mannion, and Leszek A. Maciaszek (Eds.). SCITEPRESS, 406–413. doi:10.5220/0012631600003687

[35] Yanming Yang, Xin Xia, David Lo, and John C. Grundy. 2022. A Survey on Deep Learning for Software Engineering. *ACM Comput. Surv.* 54, 10s (2022), 206:1–206:73. doi:10.1145/3505243