https://doi.org/10.36315/2024v1end123
Education and New Developments 2024

AN ANALYSIS OF GENERATIVE ARTIFICIAL INTELLIGENCE TOOLS USAGE TO ADAPT AND ENRICH SOFTWARE DEVELOPMENT COURSES

Branko Mihaljević[1], Aleksander Radovan[2], & Martin Žagar[1]
[1]RIT Croatia (Croatia)
[2]Algebra University College (Croatia)

Abstract

The adoption of generative artificial intelligence (GAI) tools has experienced rapid growth and widespread usage, significantly impacting various industries, including software development. Recently, various AI tools and services, commonly based on large language models, have demonstrated significant potential in automatic code generation, code completion, and test case generation, but also more complex refactoring, reverse engineering, and code comprehension in explaining and reasoning the code. These tools work standalone or as extensions into integrated development environments (IDEs) to assist software developers and elevate programming productivity, thus revolutionizing and streamlining the entire software development process. Teaching and conducting programming and software development courses in higher education have often posed various challenges, especially in an online environment with students with insufficient programming experience. In recent years, various education programs in computing, computer science, software engineering, and software development have faced many challenges and have taken diverse approaches to solve them, particularly related to academic integrity and plagiarism detection. However, the recent wide availability and omnipresence of GAI tools and services that are easily used by students for various tasks in software development in an uncontrolled environment outside of a classroom brought another considerable disruption that needs to be addressed. Our research presents a preliminary analysis of the possibilities of using GAI tools and services in programming and software development courses at the undergraduate level at the university of applied sciences in Croatia. Our goal was to elucidate how these tools could enhance students' learning related to software development and motivate them to acquire better programming skills based on practical assignments from the AI-assisted learning process in an ethical and legal way. At the same time, we aimed to mitigate potential risks associated with academic honesty, specifically related to plagiarism and the unallowed use of code generation. We base our research on using these GAI tools to solve individual programming tasks and software development assignments in undergraduate courses. We analyze the applicability of these tools for various programming tasks students need to perform in a responsible way and compare their performance and the level of help they offer, as well as the correctness and accuracy of results. Finally, we summarize the findings of our preliminary analysis of examined GAI tools and services that could be used in undergraduate programming and software development courses to enrich student comprehension and help educators.

Keywords: *Artificial intelligence tools, software development, programming, course, generative artificial intelligence.*

1. Introduction

In recent years, we have witnessed the rapid growth and widespread adoption of generative artificial intelligence (GAI) tools and services, significantly impacting and reshaping various industries that use them to generate text and other textual constructs, such as programming code. Moreover, predominantly built on large language models (LLM), various GAI tools and services have demonstrated significant potential in software development and programming, more specifically in tasks such as code completion, automatic code generation, and test case generation, which aim to improve the performance of software developers and the overall speed of programming process. Furthermore, these modern AI tools are also capable of doing more complex and intricate tasks in software development, such as programming code optimization and refactoring, as well as helping software developers in comprehension by explaining and reasoning the code, regardless of it is newly generated code or legacy code. Although sharing many characteristics and functionalities, the most popular AI tools and services used as programming assistants,

such as ChatGPT, Copilot, Tabnine, and CodeWhisperer, also present differences in the way they function, how users interact with them, and how they provide results and solutions. Most currently, many software developers use these AI tools to assist them in daily programming tasks by automatically completing code, suggesting code snippets, and writing whole methods based on the prompts provided.

Within the domain of higher education, namely in software development and programming undergraduate courses, the integration of artificial intelligence technology offers exceptional prospects as well as presents non-negligible obstacles. Educators are facing more complicated situations, especially ina non-controlled online learning environments, where students, particularly those with less programming experience, may start greatly depending on these tools, sometimes without the educator's approval. The widespread accessibility of AI tools also brought new concerns to academic integrity, causing educational institutions to reassess strategies for efficiently dealing with academic dishonesty problems and plagiarism.

This study explores integrating several popular AI tools within undergraduate programming and software development course curricula at our university of applied sciences. The objective of this preliminary research was to examine how these tools can not only improve the learning process by facilitating the acquisition of practical programming abilities but also tackle ethical considerations associated with their utilization. Through an analysis of the implementation of AI tools in educational environments, we aimed to monitor the capacity of these tools to enhance educational achievements while ensuring their usage aligns with principles of academic integrity. In addition, our research explores the practical uses of AI tools in software development course assignments, examining their usefulness in assisting students with programming tasks and measuring the dependability of the solutions they offer. This preliminary analysis aims to provide insights into the utilization of GAI to improve educational practices, primarily in software development courses, which helps students prepare for the changing requirements of the technology industry and develop a solid ethical foundation for their professional growth.

2. Background

Most recently, a number of research papers and studies presented their findings on the usage of AI tools and services for assistance in programming and software development (Barke et al., 2023; Daun & Brings, 2023; Puryea Ben & Sprint Gina, 2022; Vaithilingam et al., 2022). We took a similar approach and explored the usage of the most popular AI programming assistants: ChatGPT, Copilot, Tabnine, and CodeWhisperer. We examined differences in the way students use prompts to interact with them and monitor the perceived usefulness of the solutions to programming assignments they provided. Due to its widespread application in natural language processing, ChatGPT (OpenAI, 2024), based on the GPT (Generative Pre-trained Transformer) model, is best known as conversational AI. Therefore, its aptitude for comprehending and producing text that sounds human is helpful for writing documentation and explaining programming code that calls for natural language communication. An interesting potential for writing programming code as well as debugging has also been showcased by ChatGPT versions 3 and 4. However, ChatGPT's lack of deeper integration with integrated development environments (IDEs) limits its real-time coding assistance capabilities. Copilot (GitHub, 2024), developed in cooperation with GitHub and OpenAI and recently marketed as "the most widely adopted AI developer tool", was built to integrate with contemporary IDEs, such as Visual Studio Code and JetBrains IntelliJ IDEA, and work as a collaborative (pair) programmer trained on an extensive collection of publicly available code, mainly from GitHub. It automatically provides complete lines or even entire blocks of programming code on demand, conforming to the developer's coding style in seamless integration with the development environment, showcasing capacity to make it highly potent for generating high-quality code in real time. Tabnine (Tabnine, 2024) primarily emphasizes automatic code completion by leveraging machine learning models trained on diverse codebases to anticipate and propose subsequent code completions that also dynamically adjust to the developer's coding style. It is also compatible with several IDEs (e.g., VSCode, IntelliJ, PyCharm, WebStorm, Android Studio, Eclipse), allowing it to be used effectively for different programming purposes. Tabnine's model has the capability to function both locally and in the cloud, which, in some instances, can provide a notable advantage in terms of privacy (and performance) since it eliminates the need to send code externally for generating completions. CodeWhisperer (Amazon Web Services, 2024), a product developed by Amazon Web Services (AWS), bears a resemblance to GitHub Copilot but is specifically engineered to seamlessly connect with the AWS ecosystem and provide code suggestions and evaluation services integrated with AWS's range of development tools. It offers code suggestions and critiques by leveraging a range of code samples from AWS code base and user contributions, focusing strongly on adherence to industry standards. Furthermore, it prioritizes best practices and security recommendations, providing code that not only suits the task but also complies with security requirements, a vital aspect for enterprise apps.

Although some AI tools and services can operate independently, their most significant benefits come from their incorporation into IDEs, enabling them to immediately assist and support software developers in their programming tasks, improving programming productivity and revolutionizing the entire software development process. These AI tools aim to enhance efficiency and democratize programming skills by making complex coding approaches more accessible to a wider range of programmers, including those with less experience, such as students. Integrating these AI assistants into the software development process has the ability to transform conceiving, creating, and managing software completely, which may signify a new era in code production and software development in general.

3. Methodology

Since this is the first preliminary analysis of this type that we conducted at our organization, we decided to start with a group of 46 students in two sections in the second year of the undergraduate study program who were motivated to use selected AI tools (ChatGPT, Copilot, Tabnine, and CodeWhisperer) in the course teaching data structures and algorithms. Moreover, we encouraged students to explore the possibilities of these AI tools within the different IDEs they could use within ethical and legal boundaries, primarily within Visual Studio Code and IntelliJ IDEA, although other editors and IDEs were also allowed.

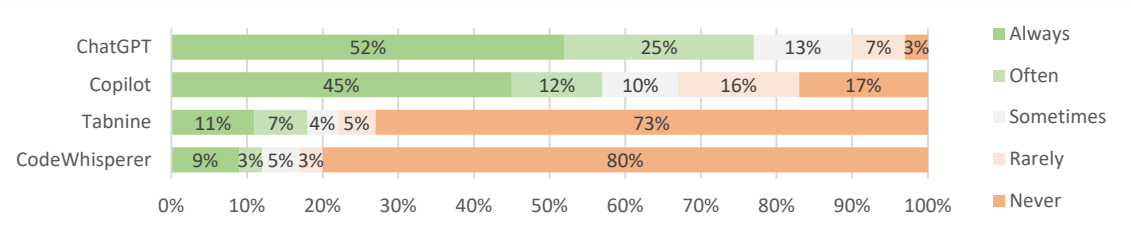
Each week, the students were presented with specific computational problems that could be solved with the assistance of AI tools and services. These problems were directly related to the topic previously covered theoretically and practically, followed by homework assignments that the students had to solve independently. More specifically, the tasks were related to the most commonly used data structures, including various types of linear data structures such as arrays, linked lists (singly, doubly, circular), stacks, queues, and decks, as well as hierarchical (non-linear) data structures such as various types of trees, maps, sets, heaps, and graphs. To solve computational problems, students had to use the data structures for which they had to create appropriate classes and methods, and implement specific algorithms. Most common computational problems introduced the implementation of the popular searching and sorting algorithms as well as recursion, backtracking, greedy, divide-and-conquer, and other techniques, with respect to time and space complexity (Big-O) and performance. Some of the problems involved the implementation of searching algorithms such as linear search, binary search, exponential search, breadth-first search (BFS), depth-first search (DFS), and other search algorithms. The other problems involved the implementation of sorting algorithms such as selection sort, bubble sort, insertion sort, merge sort, quick sort, and other variants. Some algorithms were run on specific data structures, such as Euclid's algorithm, Dijskstra's algorithm, Prim's algorithm, Kruskal's algorithm, and several others. We asked students to track their usage of AI tools in resolving all these computational problems, more precisely, how they used AI tools, how are they satisfied with their results, what challenges they faced, and what concerns they had. To achieve that, we used the anonymous survey that was each week sent to students to be responded to after successful solution submission.

The combination of close-ended questions was prepared based on prior experiences on evaluations of AI programming assistants (Barke et al., 2023; Denny et al., 2023; Moradi Dakhel et al., 2023; Puryea Ben & Sprint Gina, 2022; Vaithilingam et al., 2022). The sets of questions were prepared for the success of usage of AI tools, and challenges and concerns. The initial question (Q1) was related to the usage of offered AI assistants (ChatGPT, Copilot, Tabnine, and CodeWhisperer) throughout the semester. The group of questions (Q2-Q8) was related to students' perceived success in using AI programming assistants for specific tasks: autocompletion of code and recalling the code (Q2), generation of code with simpler programming logic (Q4), generation of repetitive code (Q4), creating code explanations and/or comments in the code (Q5), improving the efficiency and performance of the code (Q6), providing proof-of-concept and skeleton for the solution (Q7), and learning new language constructs or API/libraries (Q8). The final set of questions (Q9-Q15) was related to challenges students faced while using those AI tools and concerns they experienced, including: difficulty writing prompts and expressing requirements (Q9), relying too much on AI tools code generation (Q10), issues with understanding the code and APIs used (Q11), issues with integrating the code in the solution (Q12), concerns about the correctness and accuracy of the code (Q13), concerns about the performance of the code (Q14), and finally, concerns related to academic dishonesty, intellectual property, and plagiarism (Q15). In the next chapter, we discuss the results and our findings.

4. Results and discussion

The first question was related to the AI programming tool students used, and according to their answers, students mostly used (answers "Always" and "Often" combined) ChatGPT (77%) and Copilot (57%), while Tabnine (18%) and CodeWhisperer (12%) were used much less, as presented in Figure 1.

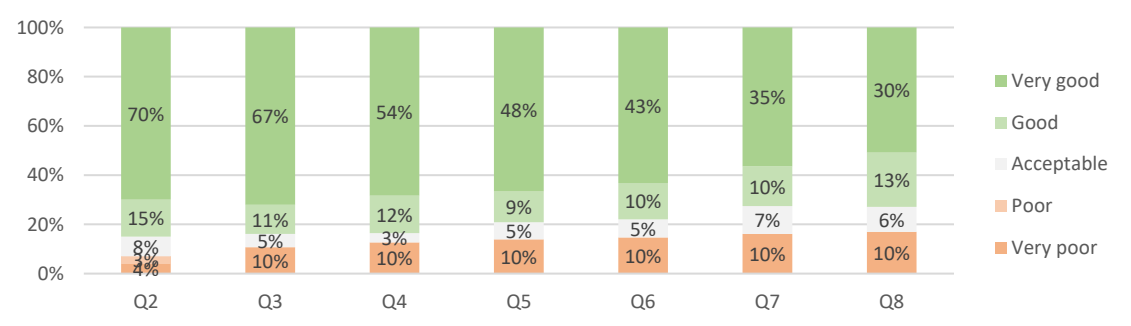
Figure 1. Usage of AI programming assistants (Q1).



The second group of questions (Q2-Q8) was related to fulfilling students' expectations and success in using AI programming assistants for specific programming tasks and activities. According to the results, students were the most successful (answers "Very good" and "Good" combined) in using AI programming assistants for the following activities:

- Autocompletion and recalling the code (Q2) was most commonly used (85%) to accelerate programming based on the standard API and libraries using IDE's autocompletion feature and recalling code syntax instead of consulting API documentation or online tutorials, or doing online search for code snippet examples (StackOverflow), which is accordance with other studies,
- Generation of code with simpler programming logic (Q3) was the second most used (78%), usually used to deal with external resources (such as reading/writing files or network resources), create independent and often static utility methods, as well as code that supports CRUD operations on typical data structures and commonly used searching and sorting algorithms,
- Generation of repetitive code (Q4) is also widely used (66%), introducing the most common functionalities that are often repeated (also referred to as "boilerplate code,"), although this was a feature already supported in some IDEs even before help from AI assistants,
- Creating code explanations and/or comments in the code (Q5) is also often successfully used (57%), thus providing an additional way of better understanding the code written by AI tools,
- Improving the efficiency and performance of the code (Q6), was used a bit less successfully (53%)
- Providing proof-of-concept and solution skeleton (Q7), was also considered less successful (45%),
- Learning new language constructs or API/libraries (Q8), was, surprisingly, considered the least successful activity (43%).

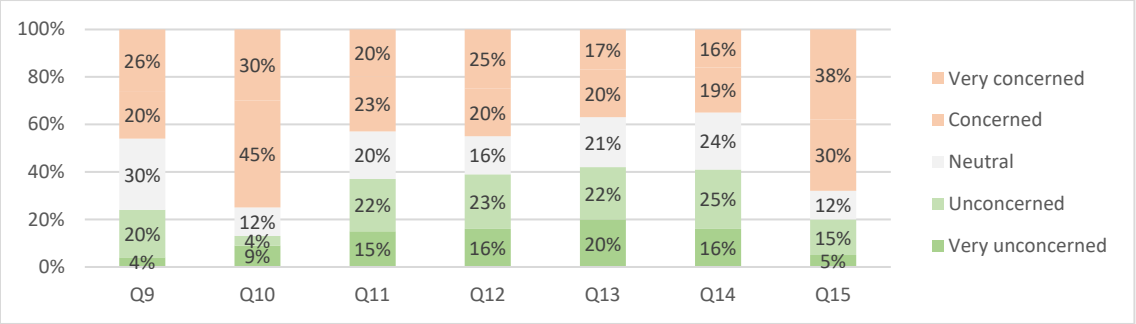
Figure 2. Perceived success using AI programming assistants for specific programming tasks and activities (Q2-Q8).



The results of the final set of questions (Q9-Q15) related to challenges students faced when using AI tools and concerns they experienced (Figure 3.), revealed significant issues students are dealing with:

- Difficulty writing prompts and expressing requirements (Q9) is considered problematic (46%),
- Relying too much on AI tools code generation (Q10) without trying to write the code themselves, which is one of the main concerns (75%) once when students got used to it,
- Issues with understanding the code and APIs used (Q11) were less significant (43%)
- Issues with integrating the code in the solution (Q12) were also present (45%)
- Concerns about the correctness and accuracy of the code (Q13) were not very large (37%),
- Concerns about the performance of the code (Q14) were even less expressed (35%)
- Concerns related to academic dishonesty, intellectual property, and plagiarism (Q15) were expressed more (68%), mostly because of a fear of how generated code would be evaluated.

Table 3. Perceived challenges and concerns when using AI tools (Q9-Q15).



5. Conclusion and future work

In this paper, we explored the practical uses of AI tools in programming and software development undergraduate course assignments, thus examining their usefulness in assisting students with programming tasks. Our preliminary findings provided insights into the utilization of AI programming assistants, primarily ChatGPT and GitHub Copilot, which students mostly used to provide autocompletion and recalling of the code, as well as generation of code with straightforward programming logic and repetitive code. On the contrary, the most significant issues students faced were related to the habit of starting to rely too much on AI tools for code generation. Our findings could be used to improve educational practices in software development courses, thus helping students better prepare for the changing requirements of the technology industry and develop a solid ethical foundation for their professional growth. Since this is preliminary research, we plan to extend it further in the next academic year with more relevant questions, additional AI tools and services for assisting students in programming and software development activities.

References

Amazon Web Services. (2024). *Amazon CodeWhisperer*. https://aws.amazon.com/codewhisperer/

Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages*, *7* (OOPSLA1). https://doi.org/10.1145/3586030

Daun, M., & Brings, J. (2023). How ChatGPT Will Change Software Engineering Education. Annual *Conference on Innovation and Technology in Computer Science Education, ITiCSE*, *1,* 110-116. https://doi.org/10.1145/3587102.3588815

Denny, P., Kumar, V., & Giacaman, N. (2023). Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. *SIGCSE 2023 - the 54th ACM Technical Symposium on Computer Science Education*, *1,* 1136-1142. https://doi.org/10.1145/3545945.3569823

GitHub. (2024). *Copilot*. https://github.com/features/copilot

Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. (Jack). (2023). GitHub Copilot AI pair programmer: Asset or Liability? *Journal of Systems and Software*, *203*, 111734. https://doi.org/10.1016/J.JSS.2023.111734

OpenAI. (2024). *ChatGPT*. https://chatgpt.com/

Puryea B., & Sprint, G. (2022). Github copilot in the classroom. *Journal of Computing Sciences in Colleges, 38*(1), 37-47. https://doi.org/10.5555/3575618.3575622

Tabnine. (2024). *Tabnine*. https://www.tabnine.com/

Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3491101.3519665