# THE EASY CHOICE IN FRONT OF THE YOUNG PROGRAMMER: HIGH PRODUCTIVITY AND ATROPHIED COMPETENCE

**Pavel Kyurkchiev, Hasan Gyulyustan, Bozhidar Samokovski**

***Abstract.*** *The mass entry of AI instruments for code generation – Github Copilot, Windsurf, Cursor, Claude Code and etc. – fundamentally change the way of learning and practised programing. For young developers these technologies demonstrate one entirely new potential to raise productivity, drastically to decrease input requirements and access to immediate applicable solutions over complex problems. That in turn gives opportunities for easier rivalry with senior developers. Despite the described benefits, the following case arises: the intensive use of helper of type (Generative AI) in early stages of learning, creates a risk of atrophy of the core competencies, required for sustainable professional growth.*

*In the current article it is analyzed the controversy between the short-term use of AI instruments for assistant programming and the negatives over critical thinking and understanding basic concepts. This will happen by an analysis of the empirical data during the European program for dual-learning and internship. This program places the young students in a professional environment with access to AI instruments and technology mentors (senior computer programmers). The research examines three core areas of competency for the degradation of young students (developers): A superficial understanding of the generated code without the ability of critical evaluation, dependency of ready-made solutions in harm of problem-solving skills and the lowered tolerance towards complexity in the presence of AI help.*

*In the Analysis it is observed that young developers that use AI instruments demonstrate lower results when dealing with complex tasks. It also establishes a correlation towards dependency of receiving ready-made solutions and motivation for adaptation towards a work environment, which includes analysis and change of already existing code.*

*In conclusion, it can be said that the only correct approach is the balanced approach, with preponderance of more complex tasks and limited access to the agents part of AI instruments. For this purpose they are offered specific pedagogical approaches that minimize risks of atrophied competence, retaining the benefits of AI instruments in the learning process.*

**Key words:** AI programming, Competencies, Education, Github Copilot, Cursor, Young Developers.

## Introduction

The appearance of AI for code generation drastically transforms the experience of the software developers [1, 2, 3]. Instruments for AI generation of code like Github Copilot, Windsurf, Cursor and Claude code lowers the confines of the possibility to generate functional code of the basis in natural language, providing easy access and drastically increase in performance. This technological revolution provides new challenges before educational institutions, as well as before professional education [4].

Project BG05SFPR001-3.001-0001 "Modernization of Vocational Education and Training", financed from the Program "The Education" 2021–2027 and co-financed from The European Union aims modernization of the curriculum and the creation of bridges between the education and the industry through inclusion of students in real work environments. During the months of June, July and August of 2025, in the frame of this project, ten students of 11th grade with "Programming" profile begin an eight-week internship representing 240 working hours in a software company. Their starting competencies include solid knowledge of object-oriented programming (OOP) with C# and experience with desktop applications development. The Task, set before the students significantly exceeds their technical capabilities: development of a corporate application with multi-layer architecture, Single page application with Back-end REST API and Front-end React website.

In the training process, the company relies on three main elements in achieving the goals: consultations with mentors (lead developers), video tutorials, and access to the AI assistant GitHub Copilot. The goals are to achieve faster development and testing of the hypothesis of whether artificial intelligence can serve as a "digital mentor" to compensate for the lack of experience and knowledge in a real work environment.

Interns are thrown into development, encouraged to use artificial intelligence to generate code, understand architectures, and solve problems while working in a team.

The scenario thus set up creates an environment for testing one of the hypotheses in modern technological education and professional development: the clash between accelerated productivity and the risk of atrophied competence. On the one hand, AI tools promise a huge increase in productivity. Young developers can deliver functionality at the speed of senior developers, creating a sense of value and success in projects [5]. On the other hand, there is a great risk that this process will completely bypass the construction of thought in students and lead to confusion in the learning process. Instead of building deep conceptual understanding, trainees may simply learn to be skilled operators of generative tools . The present study aims to explore this paradox in a real-world work environment. After examining it, it provides a framework for avoiding the traps of rapid productivity and the loss of professional competencies.

## Methodology and experimental setup

### Participants

The participants are ten students from the 11th grade (age between 16 and 17 years old) from the Mathematics High School "Academician Kiril Popov" in the city of Plovdiv, Programming profile. The internship in which the young developers participate is part of the European program "Modernization of Vocational Education and Training", which aims to promote the connection between education and business.

Each school participating in the program appoints a leader to select the best performing students, because places are limited. The duration of the program is approximately 8 weeks, spread over the months of June, July and August. Under the contract, students must be in a working environment for a duration of 240 hours of work, they must be provided with a workplace in the office of the company participating in partnership with the school. In addition to hardware, future young developers were provided with software and licenses for them. All participants have prior training in object-oriented programming with C# and experience in developing desktop applications, but declare zero or minimal experience with web technologies.

The experience was acquired as part of the curriculum for the specialty in high school education. The validation of the students' starting knowledge was verified through individual technical interviews (job interview style) by their assigned mentors in the company. The interviews strictly followed the company's templates for the technical assessment of software developers. Three assessment templates related to the project and the technology stack were selected. The templates are: Backend development with C#, Frontend development with Reach and Fundamentals.

### Project and technology stack

The task is to develop a system on the topic of "Assessing customer satisfaction", the system represents a unique framework providing the opportunity for integration with different businesses. To make this happen, a client and administrative part is developed for the system. It serves to manage and analyze the received data through weekly and monthly graphs.

The technology stack is chosen to reflect modern industrial practices:

- Backend: C# .NET 9 REST API, Entity Framework Core;
- Frontend: React 18 with TypeScript;
- Database: SQL Server 2022;
- Multi-layered architecture: The system architecture is structured into a Data Access Layer utilizing Repository and Unit of Work patterns, a Business Logic Layer with Services and Domain Models, a RESTful API Layer, and a React SPA Presentation Layer.

## Organization and workflow

The participants were divided into two teams: Team A (5 people: 2 Backend, 3 Frontend) and Team B (4 people: 2 Backend, 2 Frontend). A mentor (senior developer from the company) is assigned to each technology. The mentors from the company have the role of observers and facilitators, interfering minimally, unless the team is in a complete impasse. The organization of the workflows for creating a backlog, performing daily meetings and other tasks is guided by a third person external to the teams. The workflow completely imitates the Agile methodology with daily meetings (daily stand-ups), weekly planning and code reviews. All code is delivered to GitHub via Pull Request. Each Pull Request is directed for review to the team and the assigned mentor. The Git Flow branching strategy is fully used.

## Data collection

Quantitative data:

- Git logs: Frequency and size of commits.
- AI logs: Analysis of GitHub Copilot prompts (where possible).
- Static code analysis: Technical debt.
- Time to completion: Comparison of the 8-week deadline with historical data from the company for similar projects completed by junior specialists.

Qualitative data:

- Weekly oral defenses: Each intern defends a piece of their code in front of a mentor without the help of AI tools.

- Observations: Mentors keep a diary with observations about team dynamics, problem-solving approaches, and level of AI dependency.
- Final exam: Held in the last week, consisting of:
  - Oral interview: Repetition of the job interview conducted in the first week.
  - Practical part (60 min): Task to add new functionality to the project without access to AI tools

## Research results. The hidden cost

Quantitative results:

- Productivity: Both teams successfully delivered a working prototype of the system within 8 weeks (real work 240 hours). According to the company's data, a similar project, carried out by hired junior developers within 8 and 10 full working weeks (approximately 360 hours). This represents an acceleration of development by approximately 30–35%.
- Use of AI: Analysis of the results shows that 40% of the code in the Data Access Layer (Repository, Unit of Work, DbContext) and approximately 70% of the template code in the Frontend (components, API requests) were generated or heavily modified by AI tools. The most common prompts are of the type "How to implement Repository pattern in C# with EF Core" and "Create a React component to display a list of items". 90% reaches the generation of comments on individual methods in the code.

Qualitative results:

- Weekly oral defenses: A clear and distinct pattern is observed in the weekly interviews. Backend interns can superficially explain how a given piece of code works ("this method retrieves data from the database"), but fail when asked why ("why do we use this interface", "why do we use the Unit of work template"). Frequent answers are "the AI tool recommended it to me as a good practice" or "the AI generates it this way". The situation is even worse for Frontend interns.
- Final exam:
  - Theory: The theoretical part is an oral questioning of the company's templates. Including a questioning of the code. Here the success rate is about 45%. When offering an alternative solution, the success rate drops below 15%.

  - Practical: Only three out of nine people manage to cope with the task. All three are Backend developers, with previous experience with the technology stack.

- Mentor observations: From the observation, it becomes clear that there is no approach to fixing errors. The instinct that is created is to look for a solution with the AI tool. There is also full agreement with the suggestions made by the AI.

## The phenomenon of "Atrophied competence"

The results clearly support the hypothesis that AI is a productivity accelerator, but also creates the illusion of high knowledge and competence.

Dependency model:

- Copy-Paste-Pray: the most common problem is that code is copied and run without a thorough analysis of it.
- Piecemeal Understanding: understanding individual fragments of the code but not the entire structure and architecture.
- AI as an Oracle: AI as an absolute source of truth. It is treated like a god.

Atrophy of basic skills:

- Debugging: the most affected skill is that of finding and fixing errors. Lack of a built-in systematic approach to solving problems.
- Architectural thinking: due to receiving ready-made solutions, developers do not develop the ability to think about compromises, scalability and maintenance.
- Code reading skills: Despite the large amount of code generated, there is a loss of the ability to read and understand other people's code [6].

## Hybrid learning model

Based on the observation made, a "Hybrid Model" is proposed, which should solve the main paradox of modern learning: how to make young developers benefit from the power of AI code generation tools without atrophying fundamental problem-solving skills. The goal is to create software engineers who control the tools, not depend on them [7, 8].

## Phase "Cognitive Construction"

**Duration:** 30% of the time.

**Goal:** Building syntactic and logical independence.

In this problem, access to AI and AI tools is strictly prohibited. The goal is to build problem-solving habits and skills [9].

**Activities:**

- Writing Code: Young developers implement basic functionalities themselves to understand how they work.
- Whiteboard Coding: Solving logical problems on a piece of paper. This method involves 2 of the senses in action, which in turn helps to train the ability to plan architectures.
- Documentation instead of Prompt: Only one source of information is allowed, and this is the official documentation. This develops the search skill.

### Phase "AI as Co-pilot, not Autopilot"

**Duration:** 15% of the time.

**Objective:** To develop critical thinking.

**Pre-merge explanation:**

- Before merging, the trainee should explain what has been implemented. The process of verbal explanation ("Why does this code work?") helps the learner discover gaps in their knowledge [10].
- Implementation mentor questions.
- Prompt programming training: Examples are given on how to write commands with additional context to the AI.

### Phase "Digital detox"

**Duration:** 35% minutes of the time.

**Goal:** Maintaining brain memory and dealing with unusual situations [11].

- Deep Work: Three days a week without any AI assistants.
- Debugging sessions: Interns are given broken code. They have to find the problem and solve it [12].
- Code reading: Reading old legacy code.

### Phase "Architect and Auditor"

**Duration:** 20% of the time.

**Goal:** Integration into the real workflow [13].

- From writing to reviewing: The use of AI tools is allowed, but the emphasis is on Code Review tools, which interns should do on each other.
- Architectural Discussions: Different modules are discussed and an understanding of the system structure is emphasized [14].
- Refactoring & Cleanup: Tasks to clean up generated code.

## Conclusion

This study confirms that AI tools can reduce the development time of an application. This makes them tools with enormous potential. However, the results are not encouraging for the long-term consequences for understanding and creating code if their indiscriminate use continues to be encouraged in the organization.

The small number of participants and the short period make the study incomplete. The next steps would include expanding the scope and number of participants.

In conclusion, it can be said that the role of mentors and educational institutions is not decreasing, but rather transforming. Their role has shifted from transmitting information to cultivating critical thinking.

## References

[1] P. Denny, J. Prather et al., Computing Education in the Era of Generative AI, *Communications of the ACM*, 2024, vol. 67, no. 2, pp. 56–67, `doi:10.1145/3624720`

[2] F. Dell'Acqua et al., Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality, *SSRN Electronic Journal*, 2023, `doi:10.2139/ssrn.4573321`

[3] P. Vaithilingam, T. Zhang, E. Glassman, Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models, *CHI EA'22: CHI Conference on Human Factors in Computing Systems Extended Abstracts*, April 2022, pp. 1–7, `doi:10.1145/3491101.3519665`

[4] A. Barke, M. James, N. Polikarpova, Grounded Copilot: How programmers interact with code-generating models, *Proc. of the ACM on Programming Languages*, 2023, vol. 7, no. OOPSLA1, pp. 85–111, `doi:10.1145/3586030`

[5] N. Perry et al., Do Users Write More Insecure Code with AI Assistants?, *CCS'23: Proc. of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2785–2799, `doi:10.1145/3576915.3623157`

[6] J. Prather et al., "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers, *ACM Transactions on Computer-Human Interaction*, 2023, vol. 31, no. 1, pp. 1–31, `doi:10.1145/3617367`

[7] R. Parasuraman, D. Manzey, Complacency and Bias in Human Use of Automation: An Attentional Integration, *Human Factors The Journal of the Human Factors and Ergonomics Society*, 2010, pp. 381–410, `doi:10.1177/0018720810376055`

[8] M. Kazemitabaar et al., Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming, *CHI'23: Proc. of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, p. 1–23, `doi:10.1145/3544548.3580919`

[9] J. Sweller, Cognitive Load Theory, *The Psychology of Learning and Motivation: Cognition in Education*, 2011, vol. 55, pp. 37–76, `doi:10.1016/B978-0-12-387691-1.00002-8`

[10] M. Chi, R. Wylie, Self-explaining: The dual processes of generating inferences and repairing mental models, *Advances in Instructional Psychology*, 2000, vol. 5, pp. 161–238, ISBN: 978-0805825497

[11] E. Bjork, R. Bjork, Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning, *Psychology and the real world: Essays illustrating fundamental contributions to society*, 2011, pp. 56–64, ISBN: 978-1429230438

[12] H. Mozannar et al., Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming, *CHI'24: Proc. of the 2024 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–16, `doi:10.1145/3613904.3641936`

[13] B. Becker et al., Programming Is Hard – Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation, *SIGCSE 2023: Proc. of the 54th ACM Technical Symposium on Computer Science Education*, 2023, pp. 500–506, `doi:10.1145/3545945.3569759`

[14] V. Hellendoorn et al., When Code Completion Fails: A Case Study on Real-World Completions, *ICSE'19: Proc. of the 41st International Conference on Software Engineering*, 2019, pp. 960–970, `doi:10.1109/ICSE.2019.00101`

Pavel Kyurkchiev[1], Hasan Gyulyustan[1], Bozhidar Samokovski[1]
[1] Paisii Hilendarski University of Plovdiv,
Faculty of Mathematics and Informatics,
236 Bulgaria Blvd., 4027 Plovdiv, Bulgaria
Corresponding author: `pkyurkchiev@uni-plovdiv.bg`