Use of Programming Aids in Undergraduate Courses

Ana Rita Peixoto
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

André Glória
Instituto Universitário de Lisboa (ISCTE-IUL), Instituto de Telecomunicações, Portugal

José Luís Silva
ITI/LARSyS, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Maria Pinto-Albuquerque
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Tomás Brandão
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Luís Nunes
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Abstract

The use of external tips and applications to help with programming assignments, by novice programmers, is a double-edged sword, it can help by showing examples of problem-solving strategies, but it can also prevent learning because recognizing a good solution is not the same skill as creating one. A study was conducted during the 2nd semester of 23/24 in the course of Object Oriented Programming to help understand the impact of the programming aids in learning. The main questions that drove this study were: Which type(s) of assistance do students use when learning to program? When / where do they use it? Does it affect grades? Results, even though with a relatively small sample, seem to indicate that students who used aids have a perception of improved learning when using advice from *Colleagues*, *Copilot*-style tools, and *Large Language Models*. Results of correlating average grades with the usage of tools suggest that experience in using these tools is key for its successful use, but, contrary to students' perceptions, learning gains are marginal in the end result.

2012 ACM Subject Classification Social and professional topics → Computing education

Keywords and phrases Teaching Programming, Programming aids

Digital Object Identifier 10.4230/OASIcs.ICPEC.2024.20

Funding This work was partially supported by Fundação para a Ciência e a Tecnologia, I.P. (FCT) ISTAR Projects: UIDB/04466/2020 and UIDP/04466/2020.

1 Introduction

The possibility of using programming aids in undergraduate courses has increased gradually, but steadily, throughout the last decades, and recently was broadened with the introduction of Large Language Models (LLM). From the onset of Integrated Development Environments (IDEs), one of the available tools was the introduction of standard snippets with a hotkey, the immediate syntax highlighting of errors, or the code advisors that tried to guess the following tokens necessary to end a construct - some more successfully than others.

The use of these tools raises the question of whether students should be evaluated in their programming skills with or without these aids and also if their evaluation should include assessing their proficiency in using the available aids. On the one side, it is important to teach students to create their own solutions, on the other hand, in their future jobs, they will most certainly be able to use these and other tools to help in programming, and so it seems logical that they gather experience in using them.

The consequences reported seem to be: the change in the skills necessary to train a good programmer and the increased productivity provided by these new tools [4]. It is no longer necessary to be as meticulous and focused as before and to memorize syntax in such detail since the development environment helps with that. Also, most current languages come with extensive libraries that make algorithmic development a skill that current programmers use less frequently than a few years ago.

Code completion tools have grown more and more accurate and are evolving into code generators (as some "copilots" that have both functions integrated) [1], and Large Language Models, have contributed to increasing the sense that most programming tasks (indeed, most project development tasks) can be automated [9].

The main question, to which we will try to contribute, is whether these tools are beneficial for students learning to program or if they indeed prevent learning.

The research questions are:

- Q1: Which type(s) of assistance do students use when learning to program?
- Q2: When / where do they use it?
- Q3: Does it affect grades?

The paper contains a summarized Literature Review, a description of the experiments' Methodology and context, followed by a Results section and the corresponding Discussion, and ends with Conclusions and Future Work.

2 Literature Review

Studies regarding the use of programming aids in learning programming have been frequent in the last few years and mostly tend to conclude that the impact of code generation tools on novice programmers is significant [6]. Given the length of this work we were forced to select a few of the most obviously related.

Burak et al. [11] assessed the code generation capabilities of several code-generating tools using the benchmark HumanEval Dataset and evaluated the proposed code quality metrics. This study reveals that current tools have very different capabilities, advocating an advantage for Github Copilot.

Code completion tools assist programmers by suggesting completions for partially typed code snippets. These tools can significantly improve programming efficiency and reduce syntax errors for novice programmers. Studies such as [10] found that code completion can enhance productivity and code quality by reducing the cognitive load associated with remembering syntax and identifiers.

Kazemitabaar et al. [3] and Prather et al. [8] defend that code generators like OpenAI Codex and Github Copilot can enhance code-authoring performance and completion rates. Additionally, automated programming hints, such as next-step code hints with textual explanations, have been found to improve immediate programming performance and learning [7].

Other authors advocate that the use of code completion and code generation tools on novice programmers has significant pedagogical implications. Research by Lister et al. [5] emphasizes the importance of incorporating these tools into programming education to provide support for novices and enhance their learning experience. [2] analyses the impact of LLM generated "worked examples," concluding that students find them useful for their learning.

Automated programming hints, particularly with textual explanations and self-explanation prompts, significantly improve novice programmers' immediate performance and learning outcomes in similar subsequent tasks, according to [7]. This work reports that code hints with textual explanations significantly improved immediate programming performance. However, these hints only improved performance in a subsequent post-test task with similar objectives when they were combined with self-explanation prompts.

Code generation and completion tools automate the process of writing code. These tools can help novice programmers as they provide scaffolding and reduce the complexity of programming tasks. But, do code generation and completion tools (programming aids) enhance novice programmers' understanding of code structure and promote learning by allowing them to focus on problem-solving rather than syntax details? How do different advising strategies compare? Those are the questions we will be trying to shed some light on in the following sections.

3 Methodology

To contribute to answering the questions posed in the previous sections, we have prepared an experiment during the Object Oriented Programming (OOP) course in the first semester of 23/24 (sep-23 to dec-23). In this course, students are evaluated in class by the exercises they complete once a week in 8 of the 12 laboratories (30%), by a mid-term test (20%, a grade of less than 7.5/20 would result in immediate *Fail* in the course), and a final project with presentation and discussion (50%). Exercises and the project can be done individually or in pairs. At the beginning of the course, students were told that they could use whatever tools available to do the exercises as long as they could explain every single line of the code they delivered for evaluation, except in the mid-term test, an individual test, where no aids were allowed. During exercise evaluation and project discussion, students were frequently asked to explain and change the solutions presented so that their knowledge of what they were delivering was tested. Situations, where there were doubts concerning the students' understanding of the solution they presented, were extremely rare (2 detected cases in approximately 8 exercises $\times$ 278 students).

The students selected for this study are from three different graduation programs: Computer Science and Engineering (LEI), Computer Science and Management (LIGE), and Computer Science and Telecommunications (LETI). The first two have day and night-shifts. Night-shifts are termed "-PL" (*pós-laboral*): LEI-PL and LIGE-PL and are usually frequented by working-students. All students have a previous similar background in terms of the programming courses offered in the three programs. All programs have Introduction to Programming (one semester, 6 ECTS, mandatory course, approval is required to enroll for OOP) and an Algorithms and Data Structures course (one semester, 6 ECTS). Students enrolled in LEI have a significantly higher entrance grade.

An inquiry on the aids used was done after the grades were published to ensure that fear of influencing the grade would not be a factor. Still, the response was lower than expected (92/298 students responded, roughly 1/3 of the students enrolled in OOP).

The inquiry (originally in Portuguese, the native language of nearly all of these students) was composed of an introduction and 9 questions (the text in English of the introduction of the inquiry is in annex). The questions were the following:

1. What is your student number (optional)?
2. What is your age group? 18-23, 24-35, >35
3. Have you ever used (before OOP) tools / strategies to support learning programming (see examples in the next question)? Yes/No
4. If you answered "Yes" to the previous question, which ones? (multiple answer possible)
   - Direct advice from *colleagues*, family or friends
   - Paid *tutor* or mentoring

- Developer *communities* and online forums (StackOverflow, W3Schools...)
- Eclipse *auto-complete* (or other development environment integrated tools - IDE)
- External applications or plugins based on *Copilot* (on GitHub, InteliJ, PyCharm, ...)
- *Large Language Models*, i.e. artificial intelligence technologies for conversation (ChatGPT, Bard, ...)
- *Other*

5. Have you used any programming support tools / strategies this year in the OOP course?
   - No
   - Only in exercises
   - Only in project
   - In project and exercises
6. If you answered yes to the previous question, which ones? (options equal to question 4, multiple answer possible)
7. Which ones did you find most effective in helping to find appropriate / correct solutions? from 1 (least effective) to 5 (most effective). (options equal to question 4, but reply in [1..5] for each)
8. In your opinion, has the use of these tools improved your learning of programming? Yes/No/Don't know/Didn't use
9. If you used them, which ones did you find most useful in helping you to learn OOP? from 1 (least useful) to 5 (most useful). (options equal to question 7)

Of the 92 responses to the inquiry (out of a universe of 298 students), 44 of those have valid student identifications (question #1, inserted voluntarily), and are relatable to the students' courses and grades.

The part of the students that inserted an *id* is (naturally) biased towards more successful students, the course had a 81% success-rate but only one of the 44 responders that inserted an *id* did not pass the course (2% of the sample).

The age group of responders (question #2) was 90.2% 18-25, and 91% (84/92) claimed to have used previously some of the learning aids mentioned in the inquiry (question #3), the same number, 91%, are convinced that using these tools improved their learning ability (question #8). Only 2% (2) are convinced of the contrary.

Concerning the questions of having used aids in OOP and for what purpose (question #4), only a minority claims not to have used (14/92) (table 3).

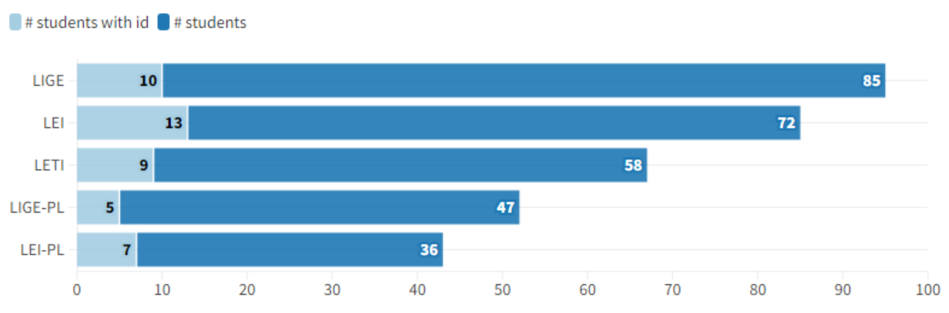

Figure 1 Number of students that responded with id and the number of students enrolled from each program.

The courses of the 44 identifiable students are: LIGE – 10/85, LETI – 9/58, LEI – 13/72, LEI-PL – 7/36, LIGE-PL – 5/47. The denominator represents the total number of students in each program enrolled in OOP. Between 10% and 20% of the population for each course responded and with a valid an *id* (figure 1).

4 Results

All types of aid seem to have been used less during OOP than previously (questions #4 and #6), according to the responses of the 92 students (figure 2). *Large Language Models (LLMs)*, *Colleagues* and *Communities* are the most admittedly used, in that order, followed by *Auto-complete*. Use of *Copilot* is low and *Tutoring* is residual (only one student in this sample admitted to have paid tutoring for OOP). Also, a low usage of *Other* (non-specified) resources. *Tutoring* and *Other* will not be considered in most of the remaining analysis due to lack of support.

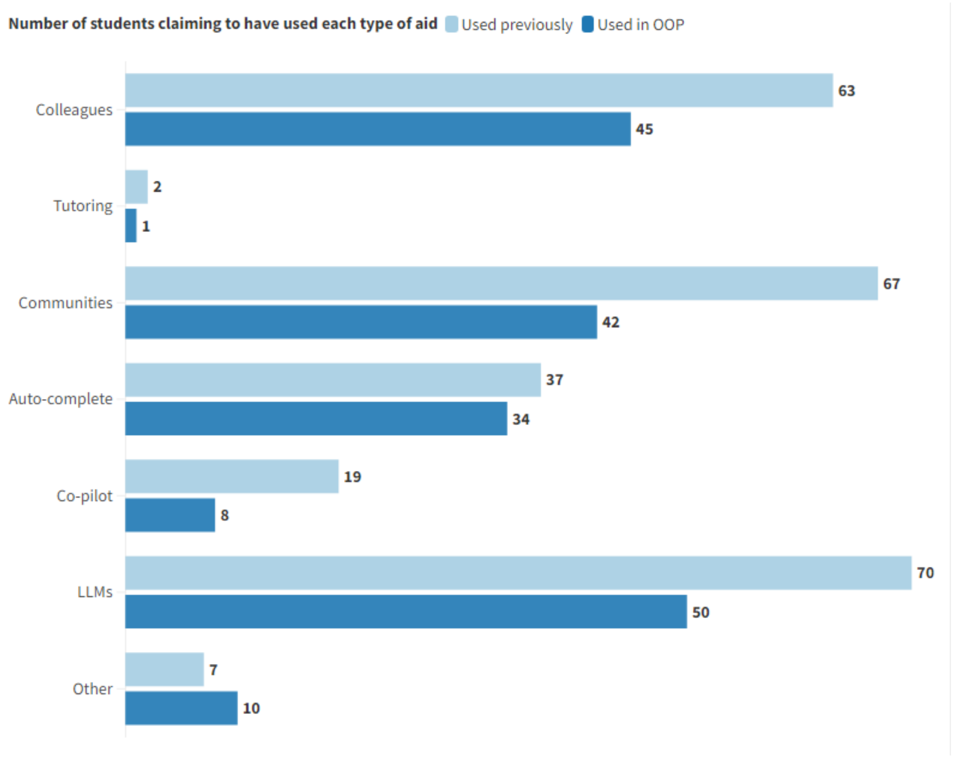

Figure 2 Histogram of declarations of the types of aid used before and during OOP.

The perceived utilities to help find solutions and learning (tables 1 and 2, questions #7 and #9, respectively) tend to be better for those that used each tool than for those that did not (with the exception of the utility of using *Copilot* to solve problems).

Favoured tools (both to find good solutions – Table 1 – and to help in the learning process – Table 2) – are unclear, although *Colleagues* have a slight advantage for solving problems, in the groups of students that actually use these tools. Also, in the group of users of the tools, *Copilot*, *Colleagues*, and *LLMs* seem to be perceived as more helpful for learning.

Students did not seem to make a difference between finding the solutions and learning with the most favoured tools. Correlations (Pearson) between votes in these categories in questions #7 and #9 are: 0.87 for *Colleagues*, 0.82 for *Copilot*, 0.73 for *Communities* and 0.73 for *LLMs*. The use of *Auto-complete* has the lowest correlation (0.59) between the perceived value for learning and for finding solutions. Other correlations (between 33 numerical variables) have no highlights apart from the obvious correlations between previous use and use in OOP of the same tools and grades of different evaluations. The variables

Table 1 Perceived utility for solving problems from those that used each tool vs those that didn't.

Type of aid	Used	Didn't use
Colleagues	3.98 +/- 0.84	3.14 +/- 1.09
Communities	3.66 +/- 0.76	3.57 +/- 1.03
Auto-complete	3.66 +/- 0.93	3.20 +/- 1.21
Copilot	3.29 +/- 1.37	4.00 +/- 1.41
LLMs	3.59 +/- 1.02	3.60 +/- 0.89

Table 2 Perceived utility for learning OOP from those that used each tool vs those that didn't.

Type of aid	Used	Didn't use
Colleagues	4.05 +/- 0.82	2.82 +/- 1.22
Communities	3.54 +/- 0.92	3.47 +/- 0.76
Auto-complete	3.47 +/- 1.07	2.72 +/- 1.19
Copilot	4.13 +/- 1.13	2.29 +/- 1.20
LLMs	3.94 +/- 0.79	2.91 +/- 1.45

correlated were the previous use of aids, the use in OOP, the types of tools used in each case, the score for effectiveness in helping to solve problems, as well as learning effectiveness and grades for each assessment.

Table 3 Counts per type of usage.

When aid was used	#count	#count students with id
No	14	6
Only in exercises	9	5
Only in final project	26	15
Exercises and final project	43	18

As for the relation between the type and frequency of aid usage with grades (Tables 3 and 4), results are unclear. The best average grade in *Exercises*, but also in the *Test* (recall, without aids) is from students that have admittedly used aids, but *only during exercises* (even though this is a relatively small sample, 5 students). What stands out in these figures is that students who admit to having used aids *only in the final project* have lower average grades in all evaluations. This is an observation with reasonable support: 15/44. Average grades of students that allegedly used *No* aids are the best in *Project* and second-best in other evaluations.

The relationship between the type of aid used and the grades (Table 5, focusing on the classes that have a reasonable number of samples, >10) appears to indicate that the type of aid used is not relevant showing only a slight decay in the average *Test* grades (the only evaluation where aids are not allowed) of students that use advice from *Colleagues* even though they have good grades in exercises.

If the *Test* results measure the true value for learning of the different types of aid, then *Communities* and *LLMs* seem to hold some learning value even though standard deviations are relatively high.

Table 4 Average grades per type of usage. Grades in [0..20].

Type of aid	Exercises	Test	Project
No	19.27 +/- 1.15	15.11 +/- 3.40	16.97 +/- 2.08
Only in exercises	19.75 +/- 0.56	16.34 +/- 3.42	16.80 +/- 2.39
Only in final project	17.16 +/- 4.45	14.45 +/- 3.55	15.07 +/- 3.90
Exercises and final project	19.03 +/- 1.59	15.09 +/- 4.92	16.47 +/- 2.61

Table 5 Average grades per type of aid used and Standard Deviations.

Type of aid	Exercises	Test	Project
Colleagues	18.79 +/- 1.57	14.42 +/- 4.68	16.09 +/- 2.56
Communities	18.35 +/- 3.10	15.86 +/- 4.43	16.52 +/- 2.69
Auto-complete	17.83 +/- 3.62	15.09 +/- 5.06	16.26 +/- 2.57
LLMs	18.02 +/- 3.52	15.77 +/- 3.66	16.03 +/- 3.45

5 Discussion

The experience of the teaching staff during this year was particularly rewarding given the reduction of retained students to $< 20\%$. The new evaluation method (based on exercises, evaluated nearly every week) is likely to have contributed to this success.

Even though we explicitly mentioned the subject at the beginning of the semester, the use of tools external to the IDE (Communities, Copilot, or ChatGPT) was seldom seen in class, where most students still seem to prefer asking colleagues or the teacher. Likely most of the aids were used during class preparation or individual work.

Apart from very few exceptions, students seemed to be quite familiar with the produced code, both in the exercises and in the final project.

The majority of the results point to a marginal difference between using aids regularly (in exercises or always) and not using any, with a slight advantage for those who used aids (the majority) even in evaluations where aids are not allowed.

Students, that used aids only in the final project, had slightly lower scores in all evaluations. The hypothesis, based on this, but that we cannot confirm with the current experiment, is that experience plays a key role in the efficient use of these tools, as in many other technological tools.

Analyzing the differences between the votes for “tools to find good solutions” and “tools that help learning,” the differences, although small, show a tendency to consider *Communities* and *Auto-complete* as tools that are more useful to “find good solutions” than to “help learning,” and *Large Language Models* the reverse.

Still, the only clue to events, that seem actually to affect grades negatively, is the use of advice from *Colleagues* that seems to have a positive effect during exercises but a negative effect during the *Test*. This may be related to the fact that being able to explain a solution is a different skill from actually being able to produce that solution.

Nevertheless, some of the automatic aids seem to affect the ability to generate solutions less than advice from *Colleagues* even though the latter should often imply the need to code the solution themselves.

6 Conclusions and Future Work

Data limitations beyond those that were expected (low participation of students) limit our conclusions, still, we believe this account presents a valid contribution to add to many others.

In this paper, we describe an experiment that took place in the 23/24 edition of the Object Oriented Programming course. In this experiment, we have explicitly liberated the use of any programming aids with the sole demand that students should (at all times) be able to explain what they were doing.

This experiment aimed at understanding which are the most usual tools used as programming aids by novice students (Q1), when / where students apply them (Q2), and how this affects their grades (Q3). The main conclusions based on the analysis of this data are the following:

- Most students use some form of programming aids (74/92, 84.8%)
- The most common types of aids used are: *Large Language Models* (50-70/92), *Colleagues* (45-63/92); *Communities* (42-67/92) and *Auto-complete* (34-37/92), in this order of preference (answering Q1). The two numbers correspond to previous use and use in OOP;
- 47% (43/92) of the students use aids in exercises and on the project, 10% (9/92) only in exercises, 28% (26/92) only use aids in the final project and 15% (14/92) claim to have used no aids (answering Q2);
- The usage of these tools does not seem to have a significant impact on the grades, neither in situations where aids are allowed nor in those where they are not (addressing Q3). However, the sample of students who claim not to have used any aids is small (n=14, 15%), and only 6 of these inserted an id, so the results lack support in this respect;
- The use of these tools for the last part of the evaluation (project) only seems to be related to students with lower grades;
- Using advice from *Colleagues* is the only event relatable to a drop of *Test* grades (the only evaluation where no aids were allowed);
- The students who reported not using any aids (although in small number) have average grades similar to those who reported using aids.

References

1 Zhamri Che Ani, Zauridah Abdul Hamid, and Nur Nazifa Zhamri. *The Recent Trends of Research on GitHub Copilot: A Systematic Review*, pages 355–366. Springer, 2024. `doi:10.1007/978-981-99-9589-9_27`.

2 Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. Evaluating llm-generated worked examples in an introductory programming course. In Nicole Herbert and Carolyn Seton, editors, *Proceedings of the 26th Australasian Computing Education Conference, ACE 2024, Sydney, NSW, Australia, 29 January 2024- 2 February 2024*, pages 77–86. ACM, January 2024. `doi:10.1145/3636243.3636252`.

3 Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. Studying the effect of AI code generators on supporting novice learners in introductory programming. In Albrecht Schmidt, Kaisa Väänänen, Tesh Goyal, Per Ola Kristensson, Anicia Peters, Stefanie Mueller, Julie R. Williamson, and Max L. Wilson, editors, *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, CHI 2023, Hamburg, Germany, April 23-28, 2023*, CHI '23, pages 455:1–455:23, New York, NY, USA, 2023. ACM. `doi:10.1145/3544548.3580919`.

4 Amy J. Ko. More than calculators: Why large language models threaten learning, teaching, and education, December 2023. URL: `https://tinyurl.com/yck47y5s`.

5 Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L. Whalley, and Christine Prasad. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In Renzo Davoli, Michael Goldweber, and Paola Salomoni, editors, *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2006, Bologna, Italy, June 26-28, 2006*, volume 38, pages 118–122, New York, NY, USA, June 2006. ACM. `doi:10.1145/1140124.1140157`.

6 Wenhan Lyu, Yimeng Wang, Tingting Chung, Yifan Sun, and Yixuan Zhang. Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study. *CoRR*, abs/2404.13414, April 2024. `doi:10.48550/arXiv.2404.13414`.

7 Samiha Marwan, Joseph Jay Williams, and Thomas W. Price. An evaluation of the impact of automated programming hints on performance and learning. In Robert McCartney, Andrew Petersen, Anthony V. Robins, and Adon Moskal, editors, *Proceedings of the 2019 ACM Conference on International Computing Education Research, ICER 2019, Toronto, ON, Canada, August 12-14, 2019*, ICER '19, pages 61–70, New York, NY, USA, 2019. ACM. `doi:10.1145/3291279.3339420`.

8 James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett B. Powell, James Finnie-Ansley, and Eddie Antonio Santos. "it's weird that it knows what I want": Usability and interactions with copilot for novice programmers. *ACM Trans. Comput. Hum. Interact.*, 31(1):4:1–4:31, November 2024. `doi:10.1145/3617367`.

9 Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *CoRR*, abs/2307.07924, July 2023. `doi:10.48550/arXiv.2307.07924`.

10 Martin P. Robillard, Wesley Coelho, and Gail C. Murphy. How effective developers investigate source code: An exploratory study. *IEEE Trans. Software Eng.*, 30(12):889–903, 2004. `doi:10.1109/TSE.2004.101`.

11 Burak Yetistiren, Isik Özsoy, Miray Ayerdem, and Eray Tüzün. Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. *CoRR*, abs/2304.10778, 2023. `doi:10.48550/arXiv.2304.10778`.

A Inquiry

With this survey, we want to study the impact of using automatic learning support tools in programming. Since the introduction of tools such as ChatGPT, the use of programming support tools has been discussed, although they have long been used in development environments, they have taken on a different proportion.

It is arguable that their use can benefit or hinder learning. That's why we set out to study their impact. This survey has only been sent out now, after the grades have been published so that there is no doubt about the possibility of this survey influencing the grades. We can also guarantee that nothing will be published that would allow participants to be identified, so I ask everyone to be as honest and thorough as possible in answering all the questions.

This survey should take no more than 5 minutes to complete.

If you have any questions about this survey, please contact: luis.nunes@iscte-iul.pt

– Place for the 9 questions of the Inquiry already presented on pages 3 and 4.