Automation of checking student assignments in IT-related subjects based on AI systems

Oleksandr A. Sharyhin[1,2], Oksana V. Klochko[3]

[1]*Dragomanov Ukrainian State University, 9 Pyrohova Str., Kyiv, 01601, Ukraine*
[2]*Miratech, 6z Vatslav Havel Blvd., Kyiv, 03124, Ukraine*
[3]*Vinnytsia Mykhailo Kotsiubynskyi State Pedagogical University, 32 Ostrozhskoho Str., Vinnytsia, 21100, Ukraine*

Abstract

Automation of students' program code verification is an important task, as it provides an opportunity to provide prompt and effective feedback to the student, significantly reducing resource costs for checking solutions. In connection with the rapid development of AI systems, new opportunities and approaches to automation appear. We consider a fundamentally new approach for the estimation of the time complexity of an algorithm. AI-based approaches compared to traditional systems that use simulations. The process of determining the complexity of the algorithm using AI-based approaches takes much less time. The study used AI systems to estimate the complexity of the algorithm based on code fragments. According to the results obtained, a decision is made regarding the suitability of these systems for automating the evaluation of students' program codes. We also offer for consideration the methods of implementing such approaches to the automation of checking student assignments in IT-related subjects based on AI systems. To assess the time complexity of code fragments, we used ChatGPT, Bard, TimeComplexity.ai, Chatsonic. All AI systems which participated in the experiment accurately determined the algorithmic complexity for each of the code fragments written in Python. The results indicate that ChatGPT and Google Bard demonstrated satisfactory accuracy in assessing the time complexity of code fragments written in Java. We developed an API that allows to partial automate teacher's work during checks of students' assignments. Further research will relate to integration of developed API into existing educational platforms and frameworks. Another area of future research is the issues of automated code quality determination and plagiarism checking.

Keywords

automation of tasks checking, AI systems, time complexity, time complexity estimation, ChatGPT, IT-related subjects, students' work, AI in education, automation in education

1. Introduction

In the dynamic realm of Information Technology (IT), the demand for skilled professionals continues to surge. While the influx of learners in IT-related disciplines is one aspect, the growing complexity and rapid evolution of the IT landscape present an equally formidable challenge for teachers [1, 2, 3]. This article advocates for a pivotal solution that could alleviate some of these challenges – the automation of student work checks in disciplines closely tied to IT.

As the IT domain evolves at an unprecedented pace, educators find themselves not only responsible for imparting foundational knowledge but also for staying abreast of the latest technological advancements. In this context, the automation of routine aspects of student work evaluation can offer a transformative opportunity [4]. By leveraging technology to handle the repetitive and time-consuming task of grading, educators specializing in IT-related disciplines can reclaim precious time. This reclaimed time, in turn, can be redirected towards more strategic and impactful activities, such as keeping pace with the swiftly evolving IT landscape, exploring innovative teaching methodologies, or engaging in research to contribute to the forefront of IT knowledge.

3L-Person 2024: IX International Workshop on Professional Retraining and Life-Long Learning using ICT: Person-oriented Approach, co-located with the 19th International Conference on ICT in Education, Research, and Industrial Applications (ICTERI 2024) September 23, 2024, Lviv, Ukraine

exhaustic@gmail.com (Oleksandr A. Sharyhin); klochkoob@gmail.com (Oksana V. Klochko)
https://scholar.google.com.ua/citations?user=69kPHxsAAAAJ (Oleksandr A. Sharyhin);
https://library.vspu.edu.ua/inform/nauk_profil.htm#klochko_oksana (Oksana V. Klochko)
0009-0006-9405-6997 (Oleksandr A. Sharyhin); 0000-0002-6505-9455 (Oksana V. Klochko)

In the study [5] the experience of automation and the use of artificial intelligence in the development of educational technologies in Sweden is considered. There are two main points about the automation of parts of a teacher's work – is the human factor problematic or desirable? So, from one point of view, human judgment can be fully or partially replaced by automation, creating a more legally secure, impartial, and fair judgment by removing emotions and uncertainties. The other argument is that core parts of teachers' work cannot be automated but also that some automation can free them from mundane routine tasks and make more time for them to do their most important work. Conversely, when people-centered skills (empathy, creativity or care) are important, then automation of these skills becomes the problem [5].

Gallagher and Breines [6] review automation in higher education and draw on a combination of events conducted with students, faculty, and staff across the three colleges at the University of Edinburgh to explore automation as a component of the teacher function assemblage.

Czibula et al. [7] note that the complexity of the algorithm, which depends on the efficiency of its work, is important for the productivity of the software system during its life cycle. Therefore, for the timely detection of system performance problems, the authors developed and automated a method for determining the complexity of the algorithm according to its execution time.

Vimalaraj et al. [8] proposed automating the marking of software code modules, analyzing it and checking it for plagiarism. Code analysis is implemented using the analysis tree.

A toolkit for checking student code written in assembly language is proposed by Liu et al. [9]. The interface of the code testing and debugging system is implemented using a browser, and an individual set of regression tests is stored for each student, before each student's code is checked using KLEE, KLC3. The system is launched when a student uploads code to a Git repository, the tutorials are implemented as LC-3, a RISC-like architecture.

The problem of students' compliance with the requirements for writing high-quality functional code remains relevant and, accordingly, the problem of its evaluation from the point of view of the implementation of a task or project is also relevant. As noted Hart et al. [10], the quality of the code itself and the quality of the style should be correlated in evaluating students' codes. To this end, they developed the Eastwood-Tidy automated listing tool for evaluating code style.

Milovancevic and Kuncak [11] developed an automated approach to verify functional programming code. The main principles of this author's approach are reliability (functional induction is used) and scalability (clustering based on the transitivity of equivalence of reference solutions).

Another approach to automated validation of student code writing tasks is to understand how readable students' code is, and whether they choose functional alternatives. These questions were investigated by Nurollahian et al. [12] using an anti-pattern in CS2 and an automatic detector.

A slightly different practice is to automatically evaluate the Arduino based on the function execution log in the virtual runtime environment for console-based languages [13]. Such a task differs in that the verification of hardware configuration (Fritzing) and source code is automated [13].

Automated code grading systems are being developed by academics to provide rapid feedback on student work. However, the problem remains that such systems do not have advanced features, including integration with Git [14]. In this study the authors developed a web-based system for automatic evaluation of C code and used GitHub and GitHub Classroom for the same purpose for comparison [14].

In addition to the program code, the comments for the code should also be of high quality [15]. This problem of software engineering is no less important than the development of quality software code. However, according to the research of the authors of this study, there are no single criteria for determining the quality of code comments, their evaluation methods are based on easily measurable attributes, binding to of a certain programming language, and specific areas, according to the research of many scientists, the evaluation of code comments is mostly implemented manually, and not automated [15].

An automated toolkit for detecting plagiarism in the source code was proposed by Rizvee et al. [16] in order to reduce the loss of resources during manual checking. Their approach to measuring the similarity between two source programs, namely string matching, uses a weighted similarity scoring

mechanism. The idea of this approach is to fix various types of plagiarism, in particular, reordering functions and changing the name of variables.

According to Klinik et al. [17], an important issue in the evaluation of students' software code is prompt feedback by students in order to improve their software solutions based on feedback. For automatic feedback generation, they developed the Personal Prof system, which implements verification based on an abstract syntactic tree of decisions and access to the semantic database of Java meta-information.

From the point of view of developing and studying the properties of test sets for automated evaluation of programs developed by students, which differ by certain criteria, for example, quality, coverage, research was conducted by Clegg et al. [18]. For this purpose, the researchers created artificial test sets of programs that simulated students' mistakes when creating software code. They proved that "different properties of test suites can influence the grades that they produce, with coverage typically making the greatest effect, and mutation score and the potentially redundant repeated coverage of lines also having a significant impact" [18].

Another direction in the automated evaluation of student-programmed problem solutions is presented by Rubio [19]. He proposed a model based on a controlled machine learning algorithm, which is used to study the sequence of partial solutions to the problem, reflecting "the programming trajectory followed by the student" [19].

Scientists have made a significant contribution to solving the problem of automated verification of students' program codes, but this problem remains relevant in the future. Modern conditions require new approaches to solving this problem. That's why we tried to solve it using artificial intelligence.

The purpose of the study is to investigate the suitability of artificial intelligence systems, which allow estimating the complexity of the algorithm based on code fragments, for automating the evaluation of students' tasks.

2. Research methods

Automation (partial or complete) of checking tasks completed by students in algorithmic and programming-related specialties has its own peculiarities. Algorithmic and programming courses often require students to demonstrate proficiency not only in coding syntax but, more critically, in designing efficient algorithms, addressing complex problem-solving scenarios, and adhering to best coding practices [20]. The multifaceted nature of these disciplines necessitates a nuanced approach to automated evaluation – one that goes beyond mere syntax checking.

Static code analyzers are special tools that perform automated and systematic checks of source code without the need for its execution. These analyzers play a major role in enhancing code quality, identifying potential issues, and enforcing coding standards. By analyzing the code structure and syntax statically, before runtime, these tools empower developers and educators alike to catch errors, ensure adherence to best practices, and improve overall code maintainability.

A well-known example of such a tool is SonarQube [21]. It supports more than 30 programming languages, detects vulnerabilities in code and checks if code follows best practices [21].

But more important (especially for courses related to algorithms and data structures) is to be able to automatically calculate code complexity.

Until recently, the main approach to estimating the complexity of an algorithm was based on running the code with a different set of input parameters. This approach is described in the study [7]. It is based on simulation and run-time measurements. This method demonstrates digestible accuracy, however even authors describe some situations when it fails.

The need for simulation, i.e. multiple runs of executable code, leads to the following disadvantages:

1. For large values, this process can take a significant amount of time, especially in the case of non-optimal algorithms or algorithms of high complexity.
2. In the case where the complexity of the algorithm is a function of two or more variables, there is a need for a larger number of simulations, which in turn also affects the time required for launches.

Thus, it can be noted that determining the complexity of an algorithm in simulation-based systems can be time-consuming.

In recent times, due to the rapid advancement of artificial intelligence systems, a fundamentally new approach to algorithm time complexity assessment has emerged, no longer relying on simulation. This approach is based on code analysis and consists of the following stages:

1. Identification of loops.
2. Analysis of operators inside loops.
3. Expression of operators as a function of "n".
4. Dominant term determination.
5. Assigning time complexity.

In this study, we will explore popular artificial intelligence systems that enable the assessment of algorithm complexity based on code fragments. These AI systems are the following:

1. ChatGPT [22].
2. Bard [23].
3. TimeComplexity.ai [24].
4. Chatsonic [25].

Building upon the obtained results, we will make decisions regarding the suitability of these systems for automating the assessment of student assignments and discuss methods for implementing this automation.

3. Results and discussion

As previously stated, we are investigating the process of automated assessment of tasks in disciplines related to algorithm design. A segment of the assessment process, pertaining to determining the complexity of the program code implementing the algorithm, is automatically conducted as a preliminary stage preceding manual evaluation by the instructor. In such a case, the flowchart representing this process would take form which is shown in figure 1.

Let us examine the AI systems that we intend to consider as suitable for assessing the time complexity of code fragments:

1. ChatGPT – one of the most well-known AI systems. It can calculate time complexity of algorithms by code fragment. In this study, we checked the two most actual versions – gpt-3.5-turbo and gpt-4. At the time of writing this work, version gpt-3.5-turbo was free, and version 4 required a monthly payment.
2. Bard – a large language system developed by Google. It can generate programming code as well as assess its time complexity.
3. TimeComplexity.ai uses the GPT-3.5 Turbo AI adapted for time complexity evaluation. It has a free and paid version.
4. Chatsonic is GPT-3 (by OpenAI) based system.

In this investigation, we examined the specified tools to assess the time complexity of code snippets written in Java and Python.

As Java examples, we used the solutions published by Nick Li for the problems presented on the LeetCode platform [26]. Below is an example of a test piece of Java code that we used to test AI systems [26]:

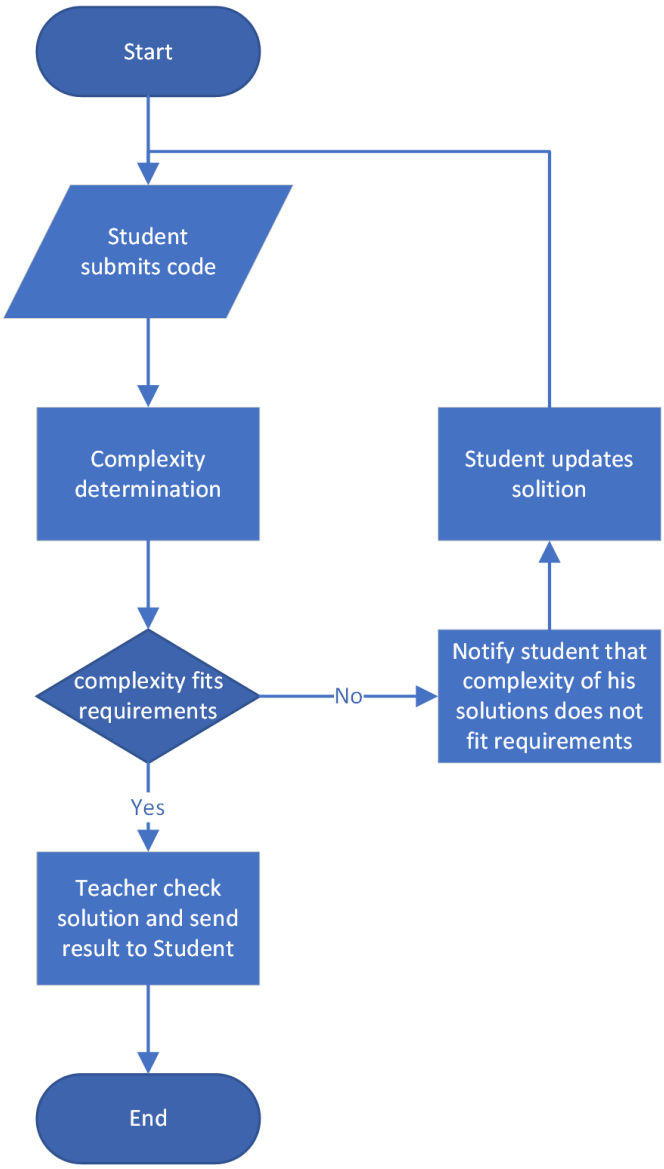

Figure 1: Tasks submission and assessment flowchart.

```
public List<List<Integer>> permute(int[] nums) {
    List<List<Integer>> result = new ArrayList<>();
    backtrack(result, new ArrayList<>(), nums);
    return result;
}

private void backtrack(List<List<Integer>> result,
List<Integer> tempList, int[] nums) {
    if (tempList.size() == nums.length) {
        result.add(new ArrayList<>(tempList));
    } else {
        for (int i = 0; i < nums.length; i++) {
          if (tempList.contains(nums[i])) continue; // element already exists,
                                                    // skip
            tempList.add(nums[i]);
            backtrack(result, tempList, nums);
```

```
            tempList.remove(tempList.size() - 1); // remove last element for next
                                                  // iteration
        }
    }
}
```

Results of executions are shown in table 1 and table 2. These tables also contain the correct value of time complexity (done by humans).

Table 1
Results of time complexity assessment of selected Java code fragments for different AI systems.

Java code fragments	gpt-3.5 turbo	gpt-4	Bard
16 : $3Sum$	n^2	n^2	n^2
18 : $4Sum$	n^3	n^3	n^3
22 : $GenerateParentheses$	$2^{2n} \cdot n$	$4^n \cdot \sqrt{n}$	2^{2n}
46 : $Permutations$	$n!$	$n \cdot n!$	$n!$
109 : $ConvertSortedList$	$n \cdot log(n)$	$n \cdot log(n)$	$n \cdot log(n)$
220 : $ConvertDupli$	$n \cdot log(k)$	$n \cdot log(min(k, n))$	$n \cdot log(k)$

Table 2
Results of time complexity assessment of selected Java code fragments for different AI systems.

Java code fragments	Time Complexity.ai	Chatsonic	Actual
16 : $3Sum$	n^2	$n \cdot log(n)$	n^2
18 : $4Sum$	n^3	$log(n) \cdot n^3$	n^3
22 : $GenerateParentheses$	$2^n \cdot n$	$2^n \cdot n$	$2^{2n} \cdot n$
46 : $Permutations$	$n!$	$n \cdot n!$	$n!$
109 : $ConvertSortedList$	n	$n \cdot log(n)$	$n \cdot log(n)$
220 : $ConvertDupli$	$n \cdot log(k)$	$n \cdot log(k)$	$n \cdot log(min(k, n))$

Our experiment involved the assessment of 32 code snippets written in the Python programming language, all AI systems which participated in the experiment accurately determined the algorithmic complexity for each of the code fragments. Code fragments in the Python programming language were taken for testing the capabilities of AI systems from student works, as well as specially developed by us as test tasks. Also each of the AI systems provided correct explanations of resulting values.

Table 1 and table 2 exclusively display instances where at least one of the systems produces an inaccurate result. In these tables, incorrect results are marked in red, and cases where the calculated value differs slightly from the correct one are marked in orange, that is, when the chain of conclusions is correct, but there are inaccuracies in the interpretation. As it follows from these tables, the Chatsonic/Writesonic system produced the highest count of inaccuracies (for “3Sum Closest”, “4Sum”, “Generate Parentheses” problems – incorrect result, for “Permutations” and “Contains Duplicate” problems – misinterpretation) in its results. This can be explained by the fact that it uses an outdated version of the GPT OpenAI API (GPT-3) as its foundational model. The use of this older API version may have impacted the AI system’s ability to accurately interpret and generate responses, leading to a higher occurrence of incorrect outputs compared to other systems under consideration.

TimeComplexity.ai exhibited inaccuracies in cases (“Generate Parentheses”, and “Convert Sorted List to Binary Search Tree” problems) where the conventional GPT-3.5 Turbo produced correct results. While this tool generally performs accurately and can be used for partial automation of verification, its limited integration capabilities, restricted to a Web UI, should be considered in practical applications.

According to the data presented, the Bard AI system demonstrated promising results, with only one instance of error detected (“Generate Parentheses” problem). This marks it as a good candidate for

integration into our research context. However, it is important to note that the API for Google Bard is currently in beta testing and is available to a limited number of users.

Gpt-3.5-turbo and gpt-4 AI systems demonstrated similar results. The only difference was for the "Contains Duplicate" problem where the gpt-4 version was able to determine the time complexity more accurately, but this difference did not significantly affect the result. We have chosen the ChatGPT AI system for our study due to its consistent and acceptable performance across various scenarios. Additionally, the availability of a public API for ChatGPT allows an efficient automation process, aligning with our research goals.

We created an API (AutoCheck API) which allows to use OpenAI functionality for automatic determination of algorithm complexity based on written code. The main method of API performs the following actions:

- connects to OpenAI;
- creates client chat completion object with own developed content under the "system" role and content of source file under "user" role;
- parses response of client chat completion.

API supports both gpt-3.5-turbo and gpt-4 versions, it can be configured by request parameters.

The simplest example which can use this API is a console application. We developed it (using Python) and used in our experiments. It performs the following actions:

- reads the content of the input file;
- calls the main method of AutoCheck API;
- redirects response it to standard output.

An example of its usage is shown in figure 2.

```
The time complexity of the above Java code is O(n log(min(n, k))), where n is the length of the input array nums[].

In the given code, an iteration is performed over the entire length of the nums array, so there's an O(n) factor.

Inside the loop, two operations are performed that could possibly affect time complexity:

1. TreeSet remove operation can take O(log n) time for the worst case. But the size of TreeSet is limited to k. So, it's
O(log min(n, k)).

2. The TreeSet ceiling operation takes O(log min(n, k)) time.

Thus, the total time complexity is O(n log(min(n, k))).

Note: TreeSet is a Red-Black tree based NavigableSet implementation in Java which is a Self-balancing Binary Search Tree.
 Hence its operation (add, remove, contains) can be performed in O(log n) time.
```

Figure 2: Example of console application output for "Contain Duplicates" problem.

Console application is just an example of how developed API can be used. It is possible to integrate it with other systems which are used in the education process – for example, Moodle. Sequence diagram of general API usage is shown in figure 3.

4. Conclusion

In this study, the process of automated assessment of tasks in disciplines related to algorithm design is addressed. It contains the step "time complexity determination" which we automate.

Various AI systems were analyzed as tools for determining the time complexity of code fragments written in the Java and Python programming languages. The results indicate that the selected systems, particularly ChatGPT and Google Bard, demonstrated satisfactory performance in accurately assessing the time complexity of code fragments written in Java and Python.

AI-based approaches are devoid of the disadvantages inherent in systems that use simulations, which leads to the fact that the process of determining the complexity of the algorithm takes much less time.

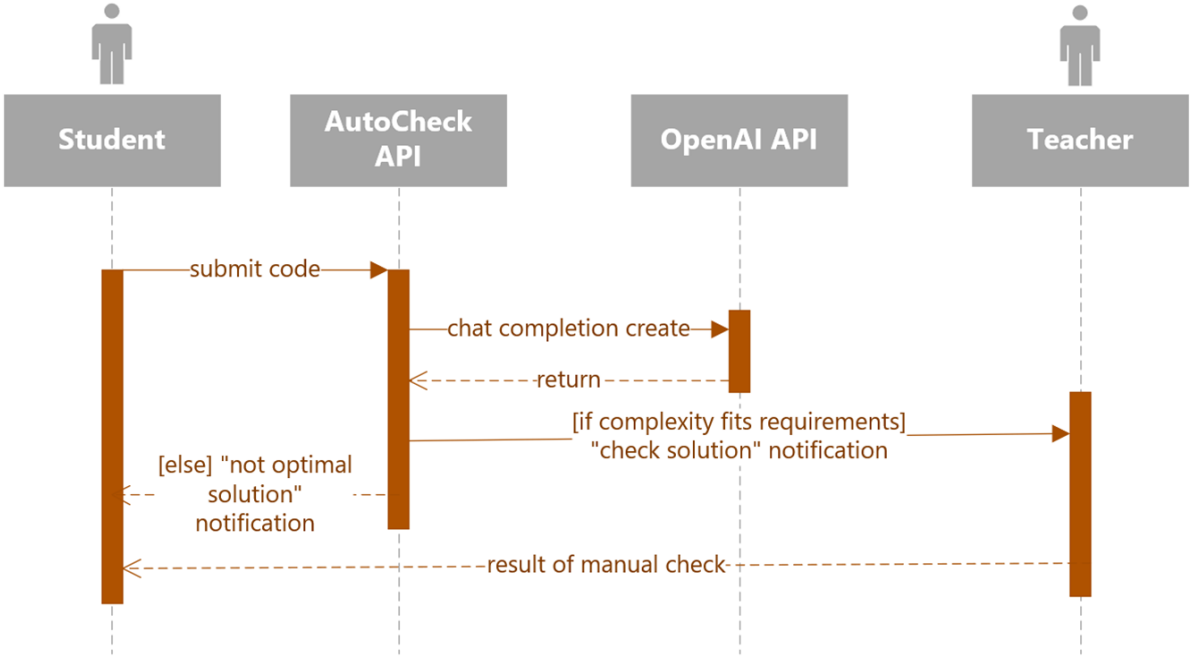

Figure 3: Tasks submission and assessment sequence diagram.

We developed an API that uses OpenAI API which allows us to use gpt-3.5-turbo and gpt-4 AI systems. Developed API allows to creation of applications that can be integrated with education platforms. Consideration of educational platforms, as well as the integration of the developed API, is an area for further research.

These results offer the opportunity for partial automation of tasks for teachers in IT-related disciplines. The time saved through automation can be redirected toward enhancing educational programs and developing creative assignments.

These findings underscore the practical applicability of AI systems in code complexity analysis, providing valuable insights for automated assessment tools in educational settings. Further research will relate to integration of developed API into existing educational platforms and frameworks.

The study examines in detail the automation of the process of determining the complexity of an algorithm, but it should be noted that the issues of automated code quality determination and plagiarism checking require further research.

References

[1] O. Y. Burov, S. H. Lytvynova, S. O. Semerikov, Y. V. Yechkalo, ICT for disaster-resilient education and training - introduction to the workshop, in: O. Y. Burov, S. H. Lytvynova, S. O. Semerikov, Y. V. Yechkalo (Eds.), Proceedings of the VII International Workshop on Professional Retraining and Life-Long Learning using ICT: Person-oriented Approach (3L-Person 2022), Virtual Event, Kryvyi Rih, Ukraine, October 25, 2022, volume 3482 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 1–25. URL: https://ceur-ws.org/Vol-3482/paper000.pdf.

[2] V. V. Osadchyi, O. P. Pinchuk, T. A. Vakaliuk, From the digital transformation strategy to the productive integration of technologies in education and training: Report 2023, in: T. A. Vakaliuk, V. V. Osadchyi, O. P. Pinchuk (Eds.), Proceedings of the 2nd Workshop on Digital Transformation of Education (DigiTransfEd 2023) co-located with 18th International Conference on ICT in Education, Research and Industrial Applications (ICTERI 2023), Ivano-Frankivsk, Ukraine, September 18-22, 2023, volume 3553 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 1–8. URL: https://ceur-ws.org/Vol-3553/paper00.pdf.

[3] O. V. Klochko, V. M. Fedorets, V. I. Klochko, K. A. Klochko, Anthropologically oriented strategies of interaction in the Human-Computer system, Journal of Physics: Conference Series 2611 (2023). doi:10.1088/1742-6596/2611/1/012018.

[4] A. L. Santos, Shifting programming education assessment from exercise outputs toward deeper comprehension, in: R. A. Peixoto de Queirós, M. P. Teixeira Pinto (Eds.), 4th International Computer Programming Education Conference (ICPEC 2023), volume 112 of *Open Access Series in Informatics (OASIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2023, pp. 1–5. doi:10.4230/OASIcs.ICPEC.2023.1.

[5] A. B. Rensfeldt, L. Rahm, Automating Teacher Work? A History of the Politics of Automation and Artificial Intelligence in Education, Postdigital Science and Education 5 (2023) 25–43. doi:10.1007/s42438-022-00344-x.

[6] M. Gallagher, M. Breines, Surfacing knowledge mobilities in higher education: reconfiguring the teacher function through automation, Learning, Media and Technology 46 (2021) 78–90. doi:10.1080/17439884.2021.1823411.

[7] I. G. Czibula, Z. Onet-Marian, R. Vida, Automatic Algorithmic Complexity Determination Using Dynamic Program Analysis, in: M. van Sinderen, L. A. Maciaszek (Eds.), Proceedings of the 14th International Conference on Software Technologies, ICSOFT 2019, Prague, Czech Republic, July 26-28, 2019, SciTePress, 2019, pp. 186–193. doi:10.5220/0007831801860193.

[8] H. Vimalaraj, T. B. K. P. Thenuwara, V. U. Wijekoon, T. Sathurjan, S. Reyal, T. A. Kuruppu, J. Tharmaseelan, Automated Programming Assignment Marking Tool, in: 2022 IEEE 7th International conference for Convergence in Technology (I2CT), 2022, pp. 1–8. doi:10.1109/I2CT54291.2022.9824339.

[9] Z. Liu, T. Liu, Q. Li, W. Luo, S. S. Lumetta, End-to-End Automation of Feedback on Student Assembly Programs, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021, pp. 18–29. doi:10.1109/ASE51524.2021.9678837.

[10] R. Hart, B. Hays, C. McMillin, E. K. Rezig, G. Rodriguez-Rivera, J. A. Turkstra, Eastwood-Tidy: C Linting for Automated Code Style Assessment in Programming Courses, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023, Association for Computing Machinery, New York, NY, USA, 2023, p. 799–805. doi:10.1145/3545945.3569817.

[11] D. Milovancevic, V. Kuncak, Proving and Disproving Equivalence of Functional Programming Assignments, Proc. ACM Program. Lang. 7 (2023) 928–951. doi:10.1145/3591258.

[12] S. Nurollahian, M. Hooper, A. Salazar, E. Wiese, Use of an Anti-Pattern in CS2: Sequential if Statements with Exclusive Conditions, in: M. Doyle, B. Stephenson, B. Dorn, L. Soh, L. Battestilli (Eds.), Proceedings of the 54th ACM Technical Symposium on Computer Science Education, Volume 1, SIGCSE 2023, Toronto, ON, Canada, March 15-18, 2023, ACM, 2023, pp. 542–548. doi:10.1145/3545945.3569744.

[13] K. Seo, J. Kim, W. J. Lee, Arduino practice judgment system based on function execution log in virtual execution environment, Comput. Appl. Eng. Educ. 32 (2024). doi:10.1002/CAE.22695.

[14] E. B. Varga, A. Kristóf Fekete, Applications for Automatic C Code Assessment, in: 2023 24th International Carpathian Control Conference (ICCC), 2023, pp. 21–26. doi:10.1109/ICCC57093.2023.10178987.

[15] P. Rani, A. Blasi, N. Stulova, S. Panichella, A. Gorla, O. Nierstrasz, A decade of code comment quality assessment: A systematic literature review, J. Syst. Softw. 195 (2023) 111515. doi:10.1016/J.JSS.2022.111515.

[16] R. A. Rizvee, M. Fahim Arefin, M. B. Abid, A Robust Objective Focused Algorithm to Detect Source Code Plagiarism, in: 2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), 2022, pp. 0109–0115. doi:10.1109/UEMCON54665.2022.9965688.

[17] M. Klinik, P. Koopman, R. van der Wal, Personal Prof: Automatic Code Review for Java Assignments, in: Proceedings of the 10th Computer Science Education Research Conference, CSERC '21, Association for Computing Machinery, New York, NY, USA, 2022, p. 31–38. doi:10.1145/3507923.3507930.

[18] B. S. Clegg, P. McMinn, G. Fraser, The Influence of Test Suite Properties on Automated Grading of Programming Exercises, in: M. Daun, E. Hochmüller, S. Krusche, B. Brügge, B. Tenbergen (Eds.), 32nd IEEE Conference on Software Engineering Education and Training, CSEE&T 2020, Virtual Conference, Germany, November 9-12, 2020, IEEE, 2020, pp. 1–10. doi:10.1109/CSEET49119.2020.9206231.

[19] M. A. Rubio, Automated Prediction of Novice Programmer Performance Using Programming Trajectories, in: I. I. Bittencourt, M. Cukurova, K. Muldner, R. Luckin, E. Millán (Eds.), Artificial Intelligence in Education, Springer International Publishing, Cham, 2020, pp. 268–272. doi:10.1007/978-3-030-52240-7_49.

[20] O. Sharyhin, V. Fedorets, O. Klochko, Monitoring and analysis of students' performance during software development, Information Technologies and Learning Tools 101 (2024) 127–149. doi:10.33407/itlt.v101i3.5586.

[21] SonarSource SA, SonarQube 10.6 Documentation, 2024. URL: https://docs.sonarsource.com/sonarqube/latest/.

[22] OpenAI, ChatGPT, 2024. URL: https://chatgpt.com/.

[23] Google, Bard, 2024. URL: https://bard.google.com/chat.

[24] J. P. Morgan, B. Brooks, TimeComplexity.ai Runtime Calculator, 2024. URL: https://www.timecomplexity.ai/.

[25] Writesonic, Chatsonic by Writesonic, 2024. URL: https://app.writesonic.com/.

[26] LeetCode, Problems, 2024. URL: https://leetcode.com/problemset/.