Adaptation of Automated Assessment System for Large Programming Courses

Marek Horváth
Department of Computers and Informatics, FEI TU of Košice, Slovakia

Tomáš Kormaník
Department of Computers and Informatics, FEI TU of Košice, Slovakia

Jaroslav Porubän
Department of Computers and Informatics, FEI TU of Košice, Slovakia

Abstract

This paper presents a new automated assessment system tailored for programming courses, addressing the challenge of evaluating a large number of students in extensive courses at the Technical University of Košice. The primary issue with current systems is their inability to handle massive course loads while ensuring objective evaluation and timely feedback. Our proposed system enhances the scalability of the assessment process, allowing for the simultaneous handling of a greater volume of assignments. It is designed to provide regular and systematic feedback to students, supporting their continuous learning and improvement. To ensure the objectivity of evaluations, the system utilizes a variety of unit test suites, selecting them randomly in each assessment to discourage students from hardcoding solutions. This approach not only supports fair and precise assessments but also significantly reduces the administrative burden on educators, enabling them to meet a wide range of educational demands.

2012 ACM Subject Classification Applied computing → Interactive learning environments; Applied computing → Computer-assisted instruction; Software and its engineering → Software creation and management

Keywords and phrases Automated Assessment, Informatics Education, Programming Feedback Systems, Continuous Integration in Education, Code Quality Analysis, Educational Technology, Computer Science Education

Digital Object Identifier 10.4230/OASIcs.ICPEC.2024.4

Funding This work was supported by project VEGA No. 1/0630/22 "Lowering Programmers' Cognitive Load Using Context-Dependent Dialogs".

1 Introduction

The expansion of programming courses across educational institutions poses significant challenges in managing the assessment of a broad spectrum of students. Traditional manual grading methods are increasingly proving to be overwhelming due to their demanding nature and potential for human error and bias.

In response, we have developed an automated assessment system tailored to manage the complexities of modern programming education, which was heavily influenced by previous findings from development of similar system [22]. This system is structured to provide continuous, regular feedback, similar to the iterative processes seen in professional software development environments where code is continuously tested and refined. This approach not only helps in the gradual enhancement of coding skills but also meets the evolving standards of industry practices.

Security and privacy in automated systems are critical, as they manage sensitive student data and are exposed to potential breaches, including threats from malicious code submissions. Therefore, our system implements essential security protocols and modules [2] to protect against these risks while maintaining high operational standards.

This paper outlines the updates to our automated assessment system, now supporting a broader spectrum of programming languages and course requirements – from basic C programming to more advanced topics such as cybersecurity in Python, along with object-oriented programming in Java. We discuss the system's scalability and flexibility, which are crucial for adapting to new testing methods and managing an increasing number of student submissions efficiently.

By detailing the system's architecture and functionality, we aim to provide insights into how to build an automated assessment system that enhances educational practices, making them more effective and responsive to the needs of both educators and students.

2 Related work

Automated assessment systems enable constant feedback for students, enhancing their learning beyond traditional methods [29][23]. These systems provide iterative feedback that helps students progressively refine their coding skills. However, the effectiveness of this feedback depends on student engagement [18].

System Arena employs parallel evaluation using Docker containers, providing detailed, data-driven insights and ensuring thorough assessments by stopping evaluations if errors are detected [19].

Code quality is crucial, preparing students for professional settings. Academic assessments often focus primarily on problem-solving rather than coding practices. Automated tools now offer context-sensitive feedback to bridge this gap [17].

Automatic evaluation advantages include speed, fairness, and objectivity, significantly improving reliability [23]. For example, JUnit tests provide specific feedback, aiding students in identifying and correcting errors [11].

Grading techniques in automated systems use formulas like:

$$\text{Final score} = (0.4 \times \text{LOC}) + (0.3 \times \text{WMC}) + (0.3 \times \text{DIT})$$

to ensure fair evaluations [1]. Threshold-based methods set clear performance standards for improvements. Automated assessments must be transparent and well-managed, ensuring a reliable framework that can adapt to the demands of programming education [9] while maintaining reasonable amount of safety [28].

Existing Tools and Technologies Used in Automated Assessment

Version control systems like Git, GitHub, and GitLab are crucial in modern programming education, facilitating functions essential for collaborative learning [27]. GitHub's management capabilities notably enhance learning outcomes and classroom efficiency [29]. Additionally, our system integrates GitLab to manage and retrieve student code submissions effectively. This integration ensures seamless communication between the assessment system and GitLab repositories, allowing for automatic synchronization and real-time feedback on student assignments.

Assessment Tools for Immediate Feedback

Advanced tools facilitate prompt and precise feedback to students, which we extensively tested during education in our department:

- **2TSW and PAAA**, **PCQL and DANTE**: These tools utilize both static and dynamic analysis to evaluate student submissions in real time [24, 26, 6, 7].

- **JavaBrat and Web-CAT**: Specifically designed to automate grading processes by analyzing both the structure and the output of code [14, 8].
- **JThreadSpy and BOSS**: These tools not only provide execution traces but also enhance the efficiency of online assignment submissions [10, 15].

These tools provide a solid foundation for effective programming education by offering feedback that is both specific and timely, thus helping students improve their coding skills. They also streamline the grading process, reducing the workload on instructors and allowing them to focus more on teaching rather than administrative tasks. Each of these tools has different approach in terms of evaluating results and similarities, which creates differences in their accuracy, performance and reliability.

3 Assessment Techniques

This section discusses the main methods used in our assessment system, explaining how static and dynamic analysis techniques help evaluate programming assignments. These methods improve the accuracy of assessments and address the needs of various programming environments.

Analysis Techniques

In programming assessment, Static Analysis and Dynamic Analysis are essential methods that work together to effectively evaluate code submissions [27]. Figure 1 shows a flowchart of these analysis techniques.

Dynamic Analysis

Dynamic analysis evaluates the actual running of programs to check their functionality against different test cases. It includes:

- Black-box testing – The internal structure of the program is unknown to the tester; the focus is solely on the inputs and outputs [3, 25, 12, 20].
- Grey-box testing – This method examines the results of each function within the program and combines these findings to assess overall performance.

Dynamic analysis is user-friendly, allowing testers to evaluate programs by comparing the actual outcomes with expected results. It can test applications without needing source code access. However, it poses risks such as security vulnerabilities, including buffer overflows that can crash servers. Dynamic analysis also relies solely on feedback from output matches and cannot assess non-compiling programs or verify compliance with specific coding instructions [16].

Static Analysis

Static analysis looks at code without running it to find potential errors and check compliance with coding standards. It includes:

- Style analysis – Evaluates the readability of code using meaningful variable names, appropriate comments, and correct indentation [1].
- Error detection – Identifies errors that might not appear during compilation but could cause runtime issues, such as infinite loops or divisions by zero.
- Metric analysis – Assesses various aspects of the program to evaluate its complexity and reliability.

- Keyword analysis – Checks for the presence of specific keywords required by the assessment criteria.
- Structural analysis – Compares the structure of the assessed program against expected solutions to judge its correctness.

Static analysis is thorough, considering all potential execution paths to identify issues. It can evaluate code with compilation errors, enhancing the depth of assessment. However, its application to complex programs is challenging due to the variety of possible solutions, which can result in misjudgments, as is proven by previously conducted research [21].

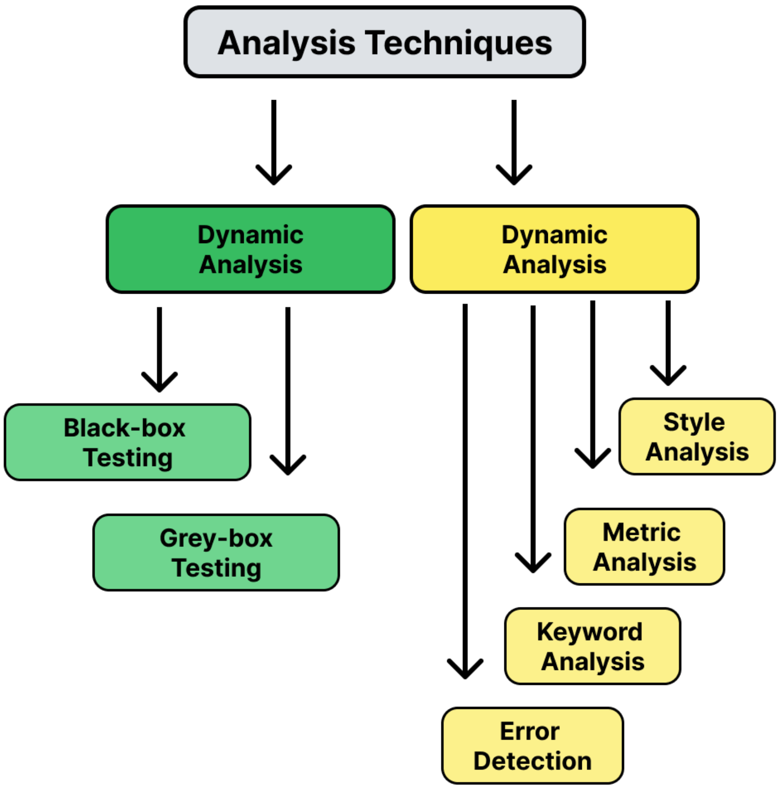


Figure 1 Flowchart illustrating analysis techniques in programming assessment.

4 Challenges and Enhancements in System Architecture for Programming Education

In the foundational course on the “Fundamentals of Programming and Algorithmization”, we currently serve approximately 1500 first-year students. Many of these students have no prior experience in programming, emphasizing the need for a system capable of providing regular feedback to facilitate learning. The course, which primarily teaches the basics of the C language, requires students to complete seven assignments throughout the semester. Given the volume of students and frequency of assignments, manually evaluating their work repeatedly is not feasible.

The complexity is further increased as these students are simultaneously enrolled in more advanced courses, such as “Object-Oriented Programming in Java”, which not only involve more complex assignments but also require significantly more system resources for evaluation.

Previously, our system relied on outdated and inflexible technologies such as SFTP for assignment submissions, which slowed down performance and hindered our ability to further improve this system. Multiple modules were affected by bad coding practices, and their versions were incompatible with currently desired technologies. Additionally, the absence of a user-friendly interface for instructors made it challenging to implement and manage new tests effectively.

To overcome these challenges, our plan includes upgrading our system to increase performance and extend its capabilities to additional programming courses such as “Data Structures”, “Operating Systems in C”, and “Cybersecurity in Python”. This enhancement will involve moving away from dependency on older systems and introducing a more interactive and manageable interface for teachers, enabling easier integration of new tests and better adaptation to evolving educational needs. This strategic upgrade is aimed at creating a more dynamic and responsive educational environment that supports both students and instructors more effectively.

In order to facilitate the development of new versions of our system, we often create either a closed group of student testers or we test results on all students while not interfering with evaluation processes in a specified subject. When the effectiveness of our system is proven and necessary issues and bugs are addressed, the system can be slowly pushed into production.

The architecture of our assessment system has been updated to better handle the increasing needs of educational environments today. This updated architecture includes high availability, automatic scaling, continuous integration and continuous deployment. Figure 2 shows how the different modules of the system interact. We’ve named each module with a callsign inspired by Roman history to make communication within our development team clearer. This distribution of systems into services is quite new since it allows us to distribute system loads across multiple physical machines. Each node in our physical machine cluster will be labeled with its specifications, and services will have their requirements defined in their manifests. This allows us to distribute tasks that require more computing power to higher-end devices or tasks that utilize CUDA to machines that are fitted with compatible GPUs.

Caesar

The **Caesar** module serves as the web application and is the primary interface for both students and teachers. It displays test results and feedback designed to help students correct their solutions. The feedback includes points awarded for various test types such as structure, static code analysis checks, error handling, and edge cases. This detailed feedback illustrates the differences between expected outputs and student submissions, aiding in identifying specific mistakes (Figure 3). For teachers, Caesar also offers functionalities to create new assignments and manage scoring, ensuring the system can handle peak access times efficiently.

Oracle

The **Oracle** module employs Python to perform data analytics on the testing and evaluation processes. It gathers statistics such as peak usage times, average success rates, code issues in individual sub-tasks and most common programming malpractices across different parts of assignments [5]. This data helps reevaluate assignment complexity and provides anonymized advice to students on effective problem-solving strategies based on the performance of the most successful peers.

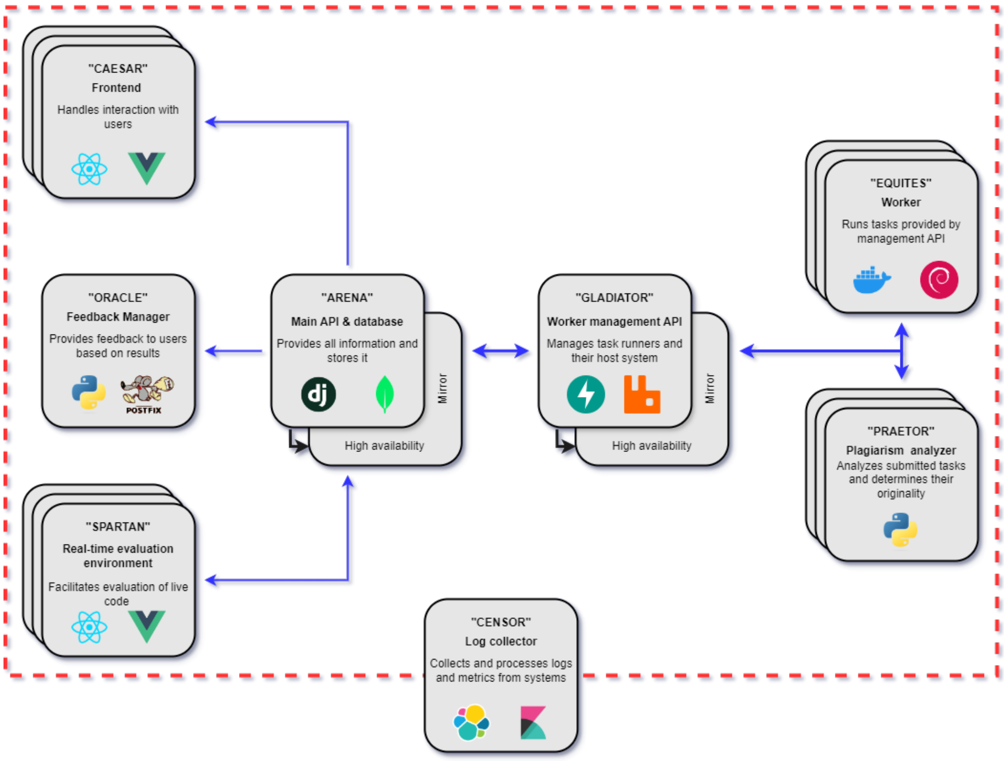
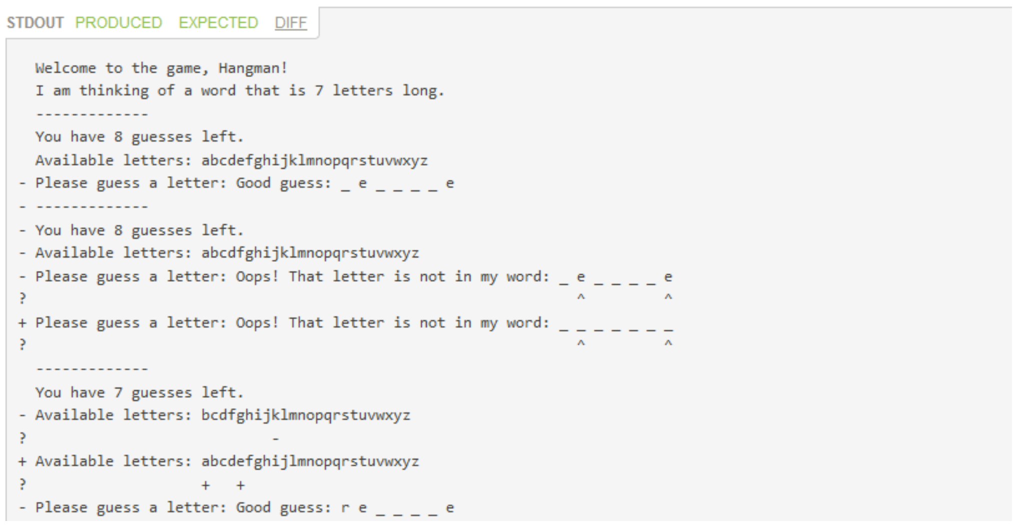

"CAESAR"
Frontend
Handles interaction with users
"ORACLE"
Feedback Manager
Provides feedback to users based on results
"ARENA"
Main API & database
Provides all information and stores it
Mirror
High availability
"GLADIATOR"
Worker management API
Manages task runners and their host system
Mirror
High availability
"EQUITES"
Worker
Runs tasks provided by management API
"PRAETOR"
Plagiarism analyzer
Analyzes submitted tasks and determines their originality
"SPARTAN"
Real-time evaluation environment
Facilitates evaluation of live code
"CENSOR"
Log collector
Collects and processes logs and metrics from systems

Figure 2 A simplified diagram of the assessment evaluation system architecture.



```
  Welcome to the game, Hangman!
  I am thinking of a word that is 7 letters long.
  -------------
  You have 8 guesses left.
  Available letters: abcdefghijklmnopqrstuvwxyz
- Please guess a letter: Good guess: _ e _ _ _ _ e
- -------------
- You have 8 guesses left.
- Available letters: abcdfghijklmnopqrstuvwxyz
- Please guess a letter: Oops! That letter is not in my word: _ e _ _ _ _ e
?                                                               ^         ^
+ Please guess a letter: Oops! That letter is not in my word: _ _ _ _ _ _ _
?                                                               ^         ^
  -------------
  You have 7 guesses left.
- Available letters: bcdfghijklmnopqrstuvwxyz
?                            -
+ Available letters: abcdefghijlmnopqrstuvwxyz
?                    +   +
- Please guess a letter: Good guess: r e _ _ _ _ e
```

Figure 3 Interface of Caesar showing detailed test feedback.

Spartan

Spartan is a complementary web application to Caesar but focuses on real-time evaluation. It provides a web-based code editor where students can write and submit code snippets for immediate feedback during class sessions. This module is particularly useful for simulating pressure-filled exam conditions and monitoring for unethical practices utilizing IP address detection and analysis of select, copy, and paste usage. We plan to further expand the features of this module in the future.

Praetor

Praetor is dedicated to plagiarism detection, employing tools like JPlag, Moss, and Sherlock, along with custom extensions for analyzing lines of code, regex patterns, and variable usage. This module is critical for maintaining academic integrity and is being developed to include a new web interface that allows teachers to actively monitor and review the plagiarism evaluation process. Previously developed experimental versions of this tool [13] have proven its effectiveness and usability while also providing valuable information for its development.

Additional Components

The **Arena** module acts as the central hub, using the Django framework and a Mongo database to facilitate communication between all modules via a RESTful API. **Gladiator** manages task distribution and system checks, utilizing FastAPI and RabbitMQ for efficient operation. **Equites**, the worker module, runs on Debian Linux and handles Docker-contained assessment processes, ensuring scalability and security. The **Censor** module, supported by Elastic and Kibana, tracks metrics and system health, aiding in predictive maintenance and machine learning applications.

This updated system architecture is designed as a robust, scalable, and secure assessment environment that adapts to the needs of both students and educators, enhancing the educational experience through technology.

5 Enhancements in Assessment Methodologies

Our recent changes and improvements have significantly enhanced the accuracy of evaluations, accelerated the assessment processes and reduced the number of false positives in plagiarism detection. On the contrary, these enhancements have led to increased consumption of system resources.

The most notable improvement has been in the infrastructure of the system. Critical updates can now be tested, organized, and deployed to production in an average of two minutes (time until the pushed update is reflected on the side of users). Optimization of used base images, elimination of unnecessary nested virtualization and cleanup of host systems have reduced resource consumption by 21% (based on normalized average statistics collected over a 30 day period).

Code quality of our system has also improved, making the code more readable and easier to maintain. Implementation of multiple linting tools and coding standards has decreased the amount of LOC by approximately 14%. We have adopted a policy for capturing all possible data processed, which could be utilized to train language models or perform predictions. During testing, we noted a visible reduction in system load (Fig. 4), particularly when adjusting the compression algorithm at the beginning and end of tests. The reduction in load during test runs is modest but could be more apparent in larger projects that use asynchronous calls and multithreading more extensively than our test samples.

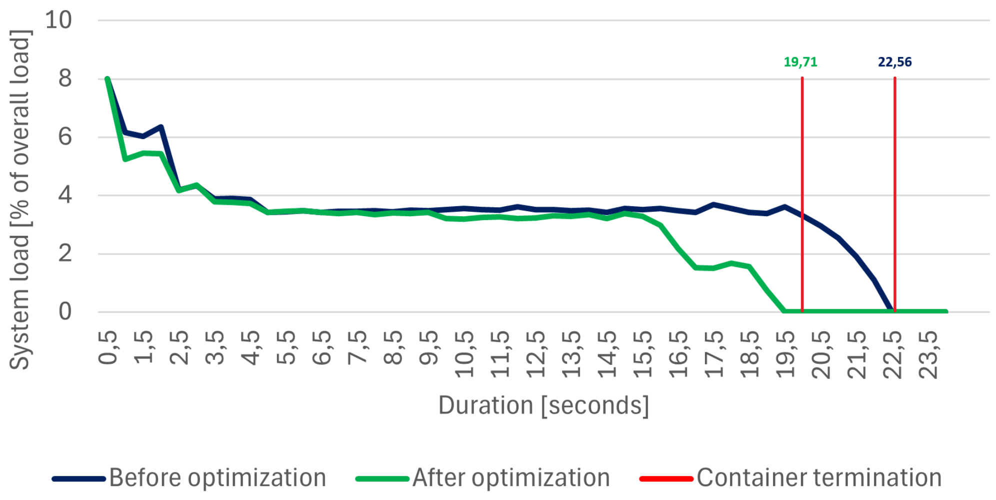


Figure 4 Comparison of container metrics based on an average of 20 runs.

We initially experimented with artificially generated scenarios for assignments. After analysis of the results, we have chosen to move away from this approach as it often produced scenarios that were too generic or oddly phrased. Qualified staff now handles the design and evaluation of each test scenario. While this approach may not be groundbreaking, it preserves the human element, addresses ethical considerations, and avoids the pitfalls associated with fully automated systems. Previously created web services [4] were already tailored for such use cases and were easily adapted to suit our needs.

The current architecture readily accommodates the evaluation of commonly used programming languages. The standard method of testing via result value comparison proved insufficiently detailed since, in some courses, not only the output of programs is considered in evaluation. A multitude of courses with more specific topics, such as "Web Technologies" or "Intelligent Systems" (a machine learning-oriented subject), require evaluation of the usability, reliability, or performance of the provided solution.

Typically, we evaluate submissions in C, Java, Python, JavaScript, and Shell. Evaluating other languages is possible but requires additional time to design appropriate tests. We plan to standardize the evaluation of Assembly source code, particularly for courses specifically focused on this language. Besides standard testing, we have also integrated basic semantic analysis of source code and are looking to expand this analysis to more effectively determine if the code was written by a human. Evaluation of non-programming assignments is possible, however we are currently focusing mainly on mentioned programming languages.

6 Future Directions

As we continue to develop our automated assessment system for programming assignments, our primary objectives include enhancing system scalability, improving usability, and expanding the range of supported programming languages. We are also exploring the integration of machine learning techniques to refine the feedback provided to students. These advancements will include analyzing code quality, identifying common programming errors, and offering customized feedback to help students enhance their coding proficiency. Major changes are also considered, for example, a change in the programming language used for heavy computing loads; Rust is a strong candidate due to its excellent performance.

Another significant area of development is expanding the system's capability to assess assignments written in various programming languages. This will not only broaden the system's applicability but also ensure its adaptability to diverse educational requirements. To further support this goal, we will incorporate advanced static analysis tools. These tools will help provide clearer feedback on code quality, helping students grasp the details of efficient coding practices. By analyzing the structural and syntactic elements of code submissions, these tools will improve the learning experience by identifying specific areas for improvement.

Finally, we plan to test the system in a production environment to evaluate its stability and performance under real-world conditions. This will include thorough usability tests with educators to ensure that the system is not only robust but also user-friendly, allowing teachers to modify assessments according to their pedagogical needs.

These enhancements are aimed at creating a more effective and responsive educational tool that supports the continuous development of both students and educators in the field of programming.

7 Conclusion

In this paper, we have presented updates to an automated assessment system designed to support programming education effectively. The system has been adjusted to manage a broader array of courses, including basics of programming, object-oriented programming, and more complex areas like cybersecurity.

Our discussions highlighted the practical applications of static and dynamic analysis in improving the accuracy of student assessments and simplifying the grading process. We have also outlined how enhancements to the system's architecture help it handle increasing student numbers and a diversity of programming courses more efficiently.

As we continue to refine this system, we are focused on making incremental improvements that support the day-to-day needs of educators and students. By sharing our experiences and the specific updates we have made, we hope to provide useful insights that can assist others in developing or enhancing their own automated assessment systems [5]. This straightforward approach aims to ensure that the system not only meets current educational demands but is also prepared to adapt to future changes in the programming education landscape.

References

1 Burcu Alper, Selma Nazlioglu, and Hurevren Kilic. Ace-pe: An automated code evaluation software tool for programming education. In *2023 11th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–5, 2023. `doi:10.1109/ISDFS58141.2023.10131776`.

2 Anton Balaz, Norbert Adam, Emilia Pietrikova, and Branislav Mados. Modsecurity idmef module. In *2018 IEEE 16th World Symposium on Applied Machine Intelligence and Informatics (SAMI 2018): Dedicated to the Memory of Pioneer of Robotics Antal (Tony) K. Bejczy*, pages 43–48. IEEE, 2018. IEEE 16th World Symposium on Applied Machine Intelligence and Informatics (SAMI) Dedicated to the Memory of Pioneer of Robotics Antal (Tony) K. Bejczy, Kosice, SLOVAKIA, FEB 07-10, 2018.

3 S. Benford, E. Burke, E. Foxley, N. Gutteridge, and A. M. Zin. Experiences with the ceilidh system. In *Proceedings of the International Conference in Computer Based Learning in Science*, 1993.

4 M. Binas. Identifying web services for automatic assessments of programming assignments. In *12th IEEE International Conference on emerging E-learning Technologies and Applications (ICETA 2014)*, pages 45–50. IEEE, 2014. 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA), Slovakia, Dec 04-05, 2014.

5 Miroslav Biňas and Emília Pietriková. Impact of virtual assistant on programming novices' performance, behavior and motivation. *Acta Electrotechnica et Informatica*, 22(1):30–36, 2022. doi:10.2478/aei-2022-0005.

6 Skanda V. C, S. S. Prasad, and G. R. Dheemanth. Assessment of quality of program based on static analysis. In *IEEE Transactions on Learning Technologies*, 2019. 2019 IEEE. doi:10.1109/T4E.2019.00072.

7 P. Duch and T. Jowrki. Dante, automated assessment of programming assignments. In *IEEE Transactions on Learning Technologies*, 2018. 2018 IEEE.

8 S. H. Edwards and M. A. Perez-Quinones. Web-cat: automatically grading programming assignments. In *Proc. Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pages 328–328, 2008.

9 M. Fabijanic, G. Dambic, B. Skracic, and M. Kolaric. Automatic evaluation of student software solutions in a virtualized environment. In *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*, pages 642–647, 2023. doi:10.23919/MIPRO57284.2023.10159927.

10 Xiang Fu, Kai Qian, Lixin Tao, and J. Liu. Apogee – automated project grading and instant feedback system for web based computing. In *SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA*, 2008. Copyright 2008 ACM.

11 Sebastian Geiss, Tim Jentzsch, Nils Wild, and Christian Plewnia. Automatic programming assessment system for a computer science bridge course - an experience report. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, pages 527–536, 2022. doi:10.1109/APSEC57359.2022.00074.

12 J. B. Hext and J. W. Winings. An automatic grading scheme for simple programming exercises. *Commun. ACM*, 12(5):272–275, May 1969.

13 Marek Horváth and Emília Pietriková. An experimental comparison of three code similarity tools on over 1,000 student projects. In *2024 IEEE 22nd World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 000423–000428, 2024. doi:10.1109/SAMI60510.2024.10432863.

14 S. Imam and V. Sarkar. Habanero-java library: a java 8 framework for multicore programming. In *Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java Platform Virtual Machines, Languages, and Tools*, pages 75–86, 2014. ACM DL.

15 Mike Joy, Nathan Griffiths, and Russell Boyatt. The boss online submission and assessment system. *J. Educ. Resour. Comput.*, 5(3):Article 2, September 2005.

16 Jan Juhar and Liberios Vokorokos. Separation of concerns and concern granularity in source code. In V Novitzka, S Korecko, and A Szakal, editors, *2015 IEEE 13th International Scientific Conference on Informatics*, pages 139–144. i'15; SSAKI KPI; KPI; Technicka University – Vkosiciach; ISVTS; IEEE, 2015. IEEE 13th International Scientific Conference on Informatics, Poprad, Slovakia, Nov 18-20, 2015.

17 Oscar Karnalim and Simon. Promoting code quality via automated feedback on student submissions. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, 2021. doi:10.1109/FIE49875.2021.9637193.

18 Christian Kaufmann, Joao Pavão, and Harald Wahl. Is there a need for automated code review to be used in teaching? : From the perspective of students. In *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, 2022. doi:10.23919/CISTI54924.2022.9820030.

19 Matej Madeja, Miroslav Biňas, and Lukáš Prokein. Continuous analysis of assignment evaluation results from automated testing platform in iterative-style programming courses. In *2019 17th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 486–492, 2019. doi:10.1109/ICETA48886.2019.9040122.

20 Urs Von Matt. Kassandra: the automatic grading system. *SIGCUE Outlook*, 22(1):26–40, January 1994.

21 Emilia Pietrikova and Sergej Chodarev. Profile-driven source code exploration. In M Ganzha, L Maciaszek, and M Paprzycki, editors, *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*, volume 5 of *ACSIS-Annals of Computer Science and Information Systems*, pages 929–934. IEEE Comp Soc; Polish Informat Proc Soc; IEEE Reg 8; IEEE Poland Sect Comp Soc Chapter; IEEE Poland Gdansk Sect Comp Soc Chapter; IEEE CIS Poland Sect Chapter; ACM Special Interest Grp Applied Comp; ACM Lodz Chapter; European Alliance Innovat; Polish Acad Sci, Comm Comp Sci; Polish Operat & Syst Res Soc; Eastern Cluster ICT Poland; Mazovia Cluster ICT, 2015. 3rd International Conference on Innovative Network Systems and Applications (iNetSApp) held in conjunction with Federated Conference on Computer Science and Information Systems (FedCSIS), Technical Univ Lodz, Lodz, Poland, SEP 13-16, 2015. `doi:10.15439/2015F238`.

22 Emilia Pietrikova, Jan Juhar, and Jana Stastna. Towards automated assessment in game-creative programming courses. In *2015 13th International Conference on emerging E-learning Technologies and Applications (ICETA)*, pages 307–312. The Amer Chamber of Commerce in the Slovak Republic; Elfa; TU; IEEE; Stu Fiit; Sanet; CTF atm; PPP; It Asociacia Slovenska; It News; EurActiv; PC revue; Education.sk; Infoware, 2015. 13th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, NOV 26-27, 2015.

23 Adam Pinter and Sandor Szenasi. Automatic analysis and evaluation of student source codes. In *2020 IEEE 20th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 000161–000166, 2020. `doi:10.1109/CINTI51262.2020.9305819`.

24 G. Polito and M. Temperini. 2tsw: Automated assessment of computer programming assignments, in a gamified web-based system. In *IEEE Transactions on Learning Technologies*, 2019. 2019 IEEE.

25 Kenneth A. Reek. The TRY system – or how to avoid testing student programs. In *Proceedings of the twentieth SIGCSE technical symposium on Computer science education, SIGCSE '89*, pages 112–116, New York, NY, USA, 1989. ACM.

26 Shao Tianyi, Kuang Yulin, Huang Yihong, and Quan Yujuan. Paaa: An implementation of programming assignments automatic assessing system. In *ICDEL 2019, May 24–27, 2019, Shanghai, China*, 2019. 2019 Association for Computing Machinery.

27 Erika Baksane Varga and Antal Kristof Fekete. Applications for automatic c code assessment. In *2023 24th International Carpathian Control Conference (ICCC)*, pages 21–26, 2023. `doi:10.1109/ICCC57093.2023.10178987`.

28 Liberios Vokorokos, Anton Balaz, and Branislav Mados. Application security through sandbox virtualization. *Acta Polytechnica Hungarica*, 12(1):83–101, 2015.

29 Soundous Zougari, Mariam Tanana, and Abdelouahid Lyhyaoui. Towards an automatic assessment system in introductory programming courses. In *2016 International Conference on Electrical and Information Technologies (ICEIT)*, pages 496–499, 2016. `doi:10.1109/EITech.2016.7519649`.