TechMecha RoboSphere | 2026, Vol. 1, No.1, 22 – 31 | ISSN Online: 3155-6019
DOI: https://doi.org/10.62718/vmca.tech-robo.1.1.SC-0126-003

OPEN ACCESS

Handwritten Terminal-Based Program Compiler and Checker

Article History:

Initial submission:	19 March 2026
First decision:	25 March 2026
Revision received:	23 May 2026
Accepted for publication:	18 June 2026
Online release:	03 July 2026

John Alex M. Gamboa[1]
Danilo R. Estrada Jr.[1]
Carl Louie A. Hernandez[1]
Clarence Q. Lacaz[1]
Tristan Kyle C. Lampa[1]
Engr. Kristine E. Pineda[2]
Engr. Ruby Rosa N. Puno[3]

[1]Student, Pampanga State University, Cabambangan, Bacolor, Pampanga, Philippines
[2]Instructor I, Pampanga State University, Cabambangan, Bacolor, Pampanga, Philippines
[3]Assistant Professor IV, Pampanga State University, Cabambangan, Bacolor, Pampanga, Philippines

Abstract

Delays, inconsistencies in evaluation, and human error are common problems associated with checking by hand programming assignments. This study thus presented an attempt in developing Handwritten Terminal-Based Program Compiler and Checker that aims at automation in assessing handwritten Java and C++ code using Optical Character Recognition (OCR) and terminal-based execution. Using a camera module and a mechanical paper feeder, the Raspberry Pi-based device scanned handwritten input, transformed it into digital text, identified logical and syntactic mistakes, and used black-box testing to verify that the output was correct. The system also included an originality check to identify potential code plagiarism and offer a structured analysis of the findings. Iterative design, testing, and project refinement were highlighted by the Agile development methodology. The OCR portion, code syntax checker, logic validator, and hardware integration were among the functional modules that went through extensive testing. The outcome demonstrated that the software modules for OCR, syntax validation, and imaging were seamlessly coordinated with the hardware components of the microcontroller, paper handling system, and imaging device. The evaluation done based on the ISO 25010 software quality model, extracted an overall weighted mean of 4.39, corresponding to a rating of 'Excellent'. Whereas data quality assessment using the ISO 25012 criteria yielded an overall weighted mean of 4.15, indicating 'Very Satisfactory'. Instructors and students alike confirmed that the system was user-friendly, reduced grading times, and produced precise and consistent results. In sum, this represented an efficient, scalable, and automated means to assess handwritten scripts within academia. By minimizing the need for manual verification, it provided greater consistency in grading, relieved some of the instructor's workload, and allowed for faster feedback to students. In turn, this shows potential to improve learning outcomes by identifying and correcting programming errors in a timely fashion. Reduced human error and faster testing cycles differentiate the system from existing automated grading tools. The successful utilization of the system demonstrates its genuine practicability as a solution for advancing assessment processes in programming education.

Keywords: Handwritten Code, OCR, Compiler, Program Checker, Automated Grading, Programming Assessment

INTRODUCTION

For a long time, the assessment of handwritten programming code has been considered a vital asset in computer engineering education, serving as a true measure of a student's fundamental understanding of coding concepts. However, this conventional method presents significant hardships for both students and educators. Instructors often spend tens of hours meticulously reviewing handwritten assignments, a process demanding constant attention to detail in syntax and logic that frequently leads to mental exhaustion and

burnout. This manual activity is not only time-consuming but also susceptible to human error and inconsistent grading standards, which can damage grading standards, which can damage grading credibility and ruin student motivation.

The disruption of the learning loop is one of the most critical consequences of manual grading. A loss of student confidence when learning new concepts often results in serious consequences for their academic progress. When students receive marked assignments long after submission, "corrective learning" fails to materialize because the "memory trace" of their original logic has already faded. As noted by Mekterović et al. (2020), the lack of objectivity and completeness in manual evaluations deteriorates as class sizes increase, making it nearly impossible to provide prompt, constructive feedback in large-scale academic environments.

Despite the prevalence of automated grading tools, a clear research gap exists most current systems are designed for typed code, failing to address the unique cognitive benefits and logistical challenges of handwritten submissions. Research by Feijóo-García et al. (2021) suggests that handwritten coding promotes better constructive learning and reasoning compared to typing, yet Zhang et al. (2023) highlight a lack of specialized, objective techniques for evaluating these manuscript works. Existing manual practices are unscalable, with many instructors reporting that evaluating a single handwritten problem can take over 20 minutes.

To bridge this gap, this study proposes the development of a "Handwritten Terminal-Based Program Compiler and Checker". This system utilizes Optical Character Recognition (OCR) – specifically, Microsoft Azure Computer Vision (Microsoft, 2023) – and terminal-based execution to automate the assessment of handwritten Java and C++ code. The study is grounded in a Theoretical and Conceptual Anchor provided by the ISO/IEC 25010 Software Quality Model and the ISO/IEC 25012 Data Quality Model, ensuring the system is evaluated for functional suitability, reliability, and data accuracy.

The significance of this study lies in its contribution to programming education by providing a scalable, objective, and efficient means of assessment. By reducing the instructor's manual workload, the system allows educators to focus on personalized student mentorship and instructional quality. Furthermore, Qian and Lehman (2019) underscore, the implementation of automated systems allows for targeted feedback that can diagnose and correct common programming misconceptions immediately, thereby enhancing student learning outcomes and ensuring a "level playing field" for all learners.

This research study aimed to develop a handwritten terminal-based code compiler and checker system for programming assignments using optical character recognition (OCR) to streamline the evaluation process and overcome inefficiencies in manual methods.

The specific objectives of the research were as follows:

1. Automate the evaluation of handwritten programming assignments to provide timely feedback for students.

2. Minimize human error and subjectivity in grading to ensure consistent and accurate evaluations.

3. Identify and address common coding errors in students' handwritten submissions to support targeted teaching strategies.

4. Apply optical character recognition (OCR) to digitize handwritten code for precise assessment.

LITERATURE REVIEW

The literature surrounding the assessment of programming highlights that manual evaluation of handwritten code is a conventional yet deeply flawed method, often leading to significant

delays and inconsistencies. Instructors frequently face immense pressure when reviewing long handwritten assignments, a task that is not only time-consuming but also highly susceptible to human error in identifying syntax and logical mistakes. Restrepo-Calle and González (2018) emphasize that manual assessment is particularly complicated because it requires the instructor to judge not only correctness but also code structure, readability, and adherence to best practices. This burden is further aggravated in large classes where, according to Mekterović et al. (2020), a lack of objectivity and completeness in evaluations often occurs as the number of submissions makes timely processing nearly impossible.

Despite these challenges, handwritten coding remains a vital educational tool. Feijóo-García et al. (2021) found that handwritten coding tests promote more constructive learning and reasoning compared to typing, as they activate different cognitive processes. To support this, Qian and Lehman (2019) underscored the necessity of targeted feedback, noting that automated systems can effectively diagnose and correct conceptual misunderstandings in introductory programming. Furthermore, Verma (2024) suggests that providing feedback on coding standards, such as naming conventions like camel case, is essential for promoting professional practices. However, transitioning from manual to automated assessment for handwritten work requires specialized techniques; Zhang et al. (2023) highlighted the need for objective evaluation methods specifically designed for manuscript (handwritten) works rather than typed submissions.

Technical implementations for such automated systems rely on technologies like Optical Character Recognition (OCR), tokenization, and Abstract Syntax Trees (AST). While open-source engines like Tesseract are widely used, they often require extensive training to handle varied handwriting styles, leading researchers to favor more robust cloud-based solutions like Microsoft Azure for higher accuracy. Evaluation integrity is often maintained through black-box testing, which Cser (2024) and Sharadin (2023) describe as a method to ensure unbiased quality assurance by focusing on program output rather than internal logic. Additionally, similarity detection using token-based matching and AST analysis is utilized to promote academic integrity in large-scale classroom settings.

Despite these advancements, a significant research gap exists in the lack of accessible, integrated hardware-software solutions that can reliably automate the entire pipeline of physical paper handling, high-accuracy OCR for varied handwriting, and terminal-based execution for handwritten code. While automated systems for typed code are well-established, there is a scarcity of scalable instruments that preserve the cognitive benefits of handwritten coding while eliminating the subjective fatigue, human error, and feedback delays inherent in manual grading. Current open-source OCR technologies still struggle with the nuances of individual penmanship, creating a need for more dependable and efficient frameworks that bridge the gap between physical student submissions and digital assessment environments.

DESIGN AND METHODOLOGY

Research Design. The Agile development methodology was utilized to allow for continuous collaboration, feedback, and iterative improvements. This approach facilitated sequential changes during the development phase, enabling the researchers to solve functional and structural problems such as mechanical paper feeding issues through short, repeated sprints rather than waiting for the project's completion. The study also incorporated a mixed methods approach, combining quantitative technical performance measurements (OCR accuracy, processing time) with qualitative assessments of system usability through surveys and interviews.

Participants and Data Sources. The evaluation involved a sample of handwritten programming assignments collected from 45 undergraduate computer engineering students (1st Year, Section C) at Don Honorio Ventura State University. Initial requirements and needs were also gauged through a survey of 11 participants to identify common issues with manual grading.

Instruments and Materials

Hardware Components: Raspberry Pi 4 (central processing), Atmega328P microcontroller (motor control), 7-inch LCD touchscreen, Raspberry Pi Camera Module 3, L293D dual H-Bridge motor driver, DC motors, buck converter step-down (12V to 5V), 16 MHz clock crystal, LED strips for internal illumination, and cooling/exhaust fans

Software Tools: Microsoft Azure Computer Vision OCR (for text recognition), Python (system logic), MySQL (data storage), Tkinter (GUI development), and Java/C++ compilers (javac and g++) for terminal=based execution

Evaluation Metrics: OCR accuracy, paper jam frequency, similarity detection rate (threshold set at 50% matching), and error-checking precision. Additionally, the system was evaluated using the ISO 25010 software quality model and the ISO 25012 data quality model

Procedures

1. ***System Development***
   - Constructed block and schematic diagrams to integrate hardware and software components.
   - Implemented a paper feeding mechanism with three stages: Paper Pickup (friction-based selection), Paper Feed (backward roll for entry), and Paper Ejection (final output to tray).
   - Configured OCR processing via Azure Computer Vision, which was selected over Tesseract due to its superior accuracy with diverse handwriting styles.

2. ***Testing Process***
   - Student handwritten codes were written on a custom template limited to 24 lines to ensure camera focus and OCR accuracy.
   - The system analyzed submissions for syntax (using text-processing), logic (using black-box output validation), and similarity (using token-based and AST-based matching).
   - Results were displayed on the LCD, including error flags and similarity percentages.

3. ***Evaluation Trials***
   - Conducted bulk testing across groups of 10, 20, 30, and 40 papers to measure mechanical reliability.
   - Calculated OCR accuracy using the formula:

$$\text{OCR Accuracy (\%)} = \frac{\text{Correctly Recognized Characters}}{\text{Total Characters in Ground Truth}} \times 100$$

   - Compared system outputs with instructor evaluations to validate accuracy and consistency.

Data Analysis. Performance was analyzed using descriptive statistics (mean, percentage accuracy, frequency counts). The Weighted Mean and Frequency were calculated from Likert scale responses (ranging from 1-Poor to 5-Excellent) to interpret user satisfaction. Reliability was assessed through repeated trials, while validity was established by comparing automated results with human grading. Plagiarism detection was evaluated through token-based and AST-based similarity measures.

Ethical Considerations. Students provided informed consent through survey participation and feedback. To maintain academic integrity, the system's similarity detection flags high-matching submissions for instructor review, ensuring that automated tools assist rather than replace human judgment in sensitive cases of potential plagiarism.

Limitations. The system currently supports only Java and C++ programming languages. It is also

restricted to processing single-sided, single-sheet submissions; multiple pages or double-sided writing are not supported by the current camera and feeder configuration. Furthermore, logic checking is limited to black-box testing (output verification) and does not analyze deeper internal code structures or non-terminal-based interactions.

RESULTS

Objective 1. Automating the Evaluation of Handwritten Programming Assignments. The system's paper feeding mechanism was tested for efficiency and reliability. As shown in Table 1, paper jams varied across test groups, with averages ranging from 2.33 (20 papers) to 9 (40 papers). The overall mean of 5.58 paper jams per test indicates that while the mechanism functioned adequately, reliability decreased with larger paper volumes.

The increase in paper jams during larger bulk tests (30 and 40 papers) suggests that the mechanical components, specifically the rollers adapted from a Canon Pixma TS207, may experience fatigue or alignment issues when processing high volumes. To mitigate damage during these incidents, an "Exit" button was implemented in the GUI, allowing the user to halt the system and manually clear the paper path. While these manual interventions ensure protection of the device, they introduce minor delays in the automated workflow. The results indicate that while the system is highly capable for small to medium class sizes, further mechanical refinement, or the use of professional flatbed scanner module, might be necessary to achieve high-volume processing.

Table 1
System Functionality Evaluation

Test Group:	Number of Exam Papers	Number of Number of Paper Jams			Average Number of Paper Jams
		Test 1	Test 2	Test 3	
A	10	5	3	2	3.33
B	20	2	2	3	2.33
C	30	10	8	5	7.67
D	40	10	10	7	9
Average Mean					5.58

Objective 2. Ensuring Consistent and Accurate Evaluations. OCR accuracy was tested using two engines: TesseractOCR and Microsoft Azure OCR. Table 2 shows that TesseractOCR achieved negligible accuracy (0.15%), even after training, while Table 3 demonstrates that Microsoft Azure OCR (Microsoft, 2023) achieved a final accuracy of 98.39%.

The failure of TesseractOCR – even after being trained with thousands of handwritten images – highlights the difficulty of local open-source engines in handling the high variability of individual handwriting styles. In contrast, Microsoft Azure's deep learning models provided near-perfect accuracy without additional training. However, even with Azure, certain characters remained problematic; for instance, the system often confused close brackets ("}") with the number "3" and semicolons (;) with the letter "I". To address this, specific handwriting guidelines were developed, such as adding a diagonal slash to the number zero ("0") to distinguish it from the letter "O". These rules are essential for maintaining the 98.39% accuracy rate required for reliable code compilation.

Table 2
TesseractOCR Trained Accuracy Testing

Test No.	Number of Handwritten Characters	Number of Characters Correctly Recognized / Not Detected	Number of Characters Not Read Correctly	TesseractOCR TRAINED Accuracy%
1	253	0	253	0%
2	273	0	273	0%
3	220	1	219	0.45%
FINAL TesseractOCR TRAINED Accuracy%				0.15%

Table 3
Microsoft Azure's OCR Reading Accuracy Testing

Test No.	Number of Handwritten Characters	Number of Characters Read Correctly	Number of Characters Not Read Correctly	Microsoft Azure's OCR Accuracy%
1	268	268	0	100%
2	238	236	2	99.16%
3	227	218	9	96%
FINAL Microsoft Azure's OCR Accuracy%				98.39%

Objective 3. Identifying and Addressing Common Errors in Student Code. The system's error detection process combined tokenization,

AST parsing, and black-box testing. Results showed that syntax and logic errors were consistently flagged, with similarity detection supporting plagiarism checks.

During a field test with 45 Computer Engineering students, the system successfully processed handwritten Java submissions and identified critical errors such as mismatched brackets and infinite loops. The similarity detection module proved effective by flagging code pairings with over 50% structural similarity, a task that is traditionally unfeasible for instructors to perform manually in large classes. Furthermore, the system closed the feedback loop by automatically emailing results to students; screenshots provided by students confirmed they received detailed reports on their compilation status and errors via Gmail. This immediate feedback is vital for correcting student misconceptions before they become habituated.

Objective 4. Evaluating System Quality and Data Integrity. System performance was assessed using ISO/IEC 25010 and ISO/IEC 25012 standards. Table 4 shows an overall weighted mean of 4.39 ("Excellent"), with high scores in performance efficiency and interaction capability. Table 5 shows a mean of 4.15 ("Very Satisfactory"), with strong ratings in accuracy, completeness, and consistency.

Table 4
Summary of the System's Overall Evaluation using ISO 25010

Evaluation Criteria	Average Weighted Mean	Descriptive Interpretation
Functional Stability	4.57	Excellent
Performance Efficiency	4.62	Excellent
Compatibility	4.48	Excellent
Interaction Capability	4.62	Excellent
Reliability	4.33	Excellent
Security	4.14	Very Satisfied
Maintainability	4.52	Excellent
Safety	3.95	Very Satisfactory
Flexibility	4.29	Excellent
Overall Weighted Mean	4.39	Excellent

The "Excellent" rating in Performance Efficiency (4.62) and Interaction Capability (4.62) reflects that the system's success in providing a responsive, user-friendly touchscreen interface for instructors. While the overall results are positive, the lower score in Safety (3.95) and Availability (3.67) indicates a vulnerability to external factors like power interruptions. To improve this, the researchers recommend the integration of an Uninterruptable Power Supply (UPS) to prevent data loss or hardware corruption during sudden outages. Overall, the data confirms that the system is highly viable and credible tool for advancing assessment processes in programming education.

Table 5
Summary of the System's Overall Evaluation using ISO 25012

Evaluation Criteria	Average Weighted Mean	Descriptive Interpretation
Accuracy	4.50	Excellent
Completeness	4.50	Excellent
Consistency	4.50	Excellent
Credibility	4.33	Excellent
Currentness	4.00	Very Satisfactory
Accessibility	4.33	Excellent
Compliance	4.00	Very Satisfactory
Confidentiality	4.00	Very Satisfactory
Efficiency	4.50	Excellent
Precision	4.17	Very Satisfactory
Traceability	3.83	Very Satisfactory
Understandability	4.17	Very Satisfactory
Availability	3.67	Very Satisfactory
Portability	4.50	Excellent
Recoverability	3.83	Very Satisfactory
Overall Weighted Mean	**4.15**	**Very Satisfactory**

DISCUSSION

The development of the Handwritten Terminal-Based Code Compiler and Checker provides an automated solution for assessing programming assignments, addressing the "logjam of pending feedback" often found in large classes. While traditional manual checking is susceptible to human error, syntax oversight, and instructor fatigue, this system ensures consistency by

applying a standardized rubric across all student submissions.

A critical technical finding in this study was the significant performance gap between different OCR engines; while TesseractOCR, even after custom training yielded a final average accuracy of only 0.15%, Microsoft Azure's OCR achieved a highly reliable accuracy of 98.39%. This disparity validates the assertions in recent literature regarding the necessity of specialized, advanced techniques for the objective evaluation of manuscript-based works as opposed to typed code submissions.

The theoretical and educational implications of this system are substantial, as it bridges the gap between traditional pedagogy and modern assessment requirements. By allowing students to continue using handwritten code which research suggests promotes better cognitive reasoning and constructive learning than typing the system preserves essential learning processes while providing the speed of digital feedback. Furthermore, the incorporation of similarity detection using Tokenization and Abstract Syntax Trees (AST) allows the system to identify structural and logical similarities. This promotes academic integrity in large-scale assessments where manual plagiarism detection is nearly impossible, thus maintaining a "level playing field" for all students. The delivery of results via Gmail ensures that students receive "targeted feedback" that is essential for correcting conceptual misunderstandings in introductory courses before the memory of the task fades.

From a practical and institutional perspective, the device demonstrates strong readiness for academic implementation, receiving a 90.9% approval rating from surveyed educators. Its high ratings under the ISO 25010 model specifically in Performance Efficiency (4.62) and Interaction Capability (4.62) confirm that the tool effectively streamlines the evaluation process without sacrificing usability. While the system achieved a "Very Satisfactory" data quality mean of 4.15 under ISO 25012, the lower ratings in Security (4.14) and Safety (3.95) highlight areas for improvement, primarily due to the lack of an integrated Uninterruptible Power Supply (UPS) to protect against data loss or hardware damage during power interruptions. By automating repetitive tasks, the system effectively alleviates instructor burnout, allowing educators to focus more on mentorship and pedagogical improvement.

Despite its successes, the study identified several technical limitations that must be addressed in future iterations. The current system is restricted to Java and C++, and its syntax detection relies on text-based processing rather than deep parsing of internal program logic. Technical constraints also exist regarding OCR accuracy for specific characters, such as the frequent misidentification of the close bracket ("}") as the number "3" and the semicolon (";") as the letter "i". To enhance the system's scholarly and practical contribution, future research should focus on expanding language support to include Python or JavaScript and integrating a flatbed scanner module to reduce inconsistencies caused by lighting and camera angles. Additionally, simulating end-user inputs to improve code coverage testing would further refine the system's reliability and its ability to provide comprehensive feedback to students.

Conclusion. This study is designed and to implement a handwritten terminal-based code compiler and checker using optical character recognition (OCR) to support the evaluation of handwritten programming activities. By means of a faster, more consistent, objective system for assessing code generated by students in Java and C++, the aim was to solve the inefficiencies and inconsistencies sometimes found in manual checking techniques.

The developed system effectively served its main purposes. It was able to precisely identify several syntax and logical mistakes including mismatched brackets, infinite loops, and incorrect outputs usually present in student code. The system significantly reduced the need for manual transcription by using OCR to translate handwritten code into machine-

readable text, thus optimizing the evaluation process. Microsoft Azure Computer Vision was specifically used to replace the previously trained Tesseract OCR engine, improving recognition reliability and accuracy across many handwriting styles.

Recommendations. The system has several limitations even though it is quite effective. Currently, it supports only Java and C++ languages, which are widely used in courses in beginner to intermediate programming courses. Furthermore, instead of deep parsing, the syntax error detection relies on simple text-based techniques. Logic checking is carried out using a black-box approach evaluating only the output of the program without examining at its internal logic. These design decisions limit the system's capacity to assess deeper programming constructs or provide thorough comments on the reasoning inside the code, even while they make the system simple and efficient.

Overall, the system demonstrates that it is both possible and advantageous to assist in the evaluation of handwritten programming assignments. It offers a standardized method for evaluation, increases accuracy, and shortens the time required for grading. Such systems may eventually support a wider variety of programming languages, provide more comprehensive feedback, and integrate seamlessly into modern learning environments with additional development.

Improve Code Coverage. A major challenge in controlling code coverage is the input that must be provided by the end-user. You can change code that depends on input with another piece of code detached and then test that the code works all together without the user input or with minimal user input. By simulating the appropriate input, we can enhance testing coverage, leading to better reliability and consistency of code execution across many more users.

Flatbed Scanner Module. To further enhance the accuracy and consistency of code extraction, we could also integrate a flatbed scanner as an alternative to or in combination with the existing camera module. Using a flatbed scanner would greatly reduce inconsistencies caused by changes in lighting and camera angles of view, which translates to improved clarity and reliability in the image captures of handwritten code.

Use an Uninterruptible Power Supply (UPS). Use an Uninterrupted Power Supply to ensure the continual operation of your software tools and keep everything safe from a power failure. This device will help maintain operation during scans and processing, safeguarding against power interruptions that may lead to lost data or corrupt hardware failures due to sudden on/off behavior.

Add Additional Programming Language Support. Consider adding more programming language support so the system in future could be better utilized in more institutions and therefore would be more flexible and applicable in various educational contexts. Adding programming language support such as Python or JavaScript will enhance the device's flexibility and applicability to a more varied curriculum.

For future use, enhance the formatting of your code and OCR features so the system can thoroughly inspect the contents of the code files and e.g. support only those lines which represent single characters, i.e., brackets, to avoid syntax errors can be avoided when running the compilations.

Author contributions. Ruby Rosa N. Puno: Conceptualization, Writing – review and editing, Supervision (Thesis Coordinator) | Kristine E. Pineda: Conceptualization, Writing – review and editing, Supervision (Thesis Adviser) | John Alex M. Gamboa: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft | Danilo R. Estrada Jr.: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration,

Resources, Software, Validation, Visualization, Writing – original draft | Carl Louie A. Hernandez: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft | Clarence Q. Lacaz: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft | Tristan Kyle C. Lampa: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft.

Conflict of interest. The authors declare no conflict of interest.

Funding source. This research received no external funding.

Artificial intelligence use. AI-assisted language editing was performed using ZeroGPT; authors reviewed and approved all contents. Such measure was pursued for grammar purposes and the restructuring of complex sentence structures. The other parts of the paper maintain full integrity in terms of originality.

Ethics approval statement. In accordance with the ethical standards of the Computer Engineering Department of the Pampanga State University, the protection of participants' rights and welfare was ensured.

Data availability statement. The dataset will be available upon request from the corresponding author of this study.

Acknowledgement. The researchers sincerely extend their gratitude to the Pampanga State University for its invaluable support in the conduct of this study. Special appreciation is given to the Computer Engineering Department for providing guidance and assistance in facilitating the survey among BSCpE faculty members and students. The cooperation of the faculty members and students was instrumental in gathering the necessary data and ensuring the integrity of the research process. Their contributions and commitment greatly enriched the findings of this study.

Publisher's disclaimer. The views expressed in this article are those of the authors and do not necessarily reflect the views of the publisher. The publisher disclaims any responsibility for errors or omissions.

REFERENCES

Cser, T. (2024). *What is black box testing? methods, types, and advantages*. AI Test Automation with Machine Learning. https://www.functionize.com/automated-testing/black-boxtesting

Feijóo-García, P. G., Chen, Y., Esmaeili, S., Ma, Y., & Gardner-McCune, C. (2021). Write2code: penbased educational tool for java. *International Journal of Emerging Technologies in Learning (iJET), 16*(03), 307. https://doi.org/10.3991/ijet.v16i03.17979

ISO 25010. (n.d.). https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

ISO/IEC 25012. (n.d.). https://iso25000.com/index.php/en/iso-25000-standards/iso-25012/136-iso-iec-25012

Mekterović, I., Brkić, L., Milašinović, B., & Baranović, M. (2020). Building a comprehensive automated programming assessment system. *IEEE Access, 8*, 81154-81172. https://doi.org/10.1109/access.2020.2990980

Microsoft. (2023). *Computer Vision documentation*. Microsoft Azure. https://learn.microsoft.com/en-us/azure/cognitive-services/computer-vision/overview

Patel, R. (2020). Tesseract OCR: An Overview and Use Cases. *Towards Data Science*. https://towardsdatascience.com/tesseract-ocr-an-overview-and-use-cases-1b03f5e6dc3b

Qian, Y. & Lehman, J. D. (2019). Using targeted feedback to address common student misconceptions in introductory programming: a data-driven approach. *Sage Open, 9*(4). https://doi.org/10.1177/2158244019885136

Restrepo-Calle, F. & González, F. A. (2018). Continuous assessment in a computer programming course supported by a software tool. *Computer Applications in Engineering Education, 27*(1), 80-89. https://doi.org/10.1002/cae.22058

Sharadin, G. (2023). *What is Black Box Testing | Techniques & Examples* | Imperva. Learning Center. https://www.imperva.com/learn/application-security/black-box-testing/

Verma, A. (2024). *Coding Standards and Best Practices to Follow* | BrowserStack. https://www.browserstack.com/guide/coding-standards-best-practices?utm_source

Zhang, T., Wang, Y., Jin, X., Gu, Z., Zhang, X., & He, B. (2023). An auto-grading oriented approach for off-line handwritten organic cyclic compound structure formulas recognition. Computer Modeling in Engineering &Amp; *Sciences, 135*(3), 2267-2285. https://doi.org/10.32604/cmes.2023.023229