Beyond Correctness: A Competency-Driven Framework for Designing Autograder Test Suites

Hugo F. Guarilha[1], Nicolas Arruda[1], Carla Delgado[1], Laura de O. F. Moraes[2]

[1] Instituto de Computação – Universidade Federal do Rio de Janeiro (UFRJ)

[2]Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{hugofg,nicolasma,carla}@ic.ufrj.br, laura@uniriotec.br

Abstract. *Automated programming autograders are essential for providing immediate feedback in programming education. However, conventional autograders are often limited to evaluating functional correctness through pass/fail tests. This article introduces a framework for designing autograder test suites where a single programming problem is deconstructed into multiple competencies. To automatically assign a grade to a student's activity, the tool allows for defining weights for each test case, supporting the instructor in designing a test suite aligned with the learning objectives related to the predefined competencies. An experiment comparing this framework with traditional paper-based evaluations revealed a 97% reduction in grading time ($r = 0.70$ correlation), while effectively shifting the instructor's role from grader to assessment designer.*

1. Introduction

The teaching of programming faces the significant challenge of providing a good user experience and instant feedback to a large number of students. To address this issue, automated assessment systems, platforms that enable instructors to present questions, and students to take exams with auto-grading systems, also called online-judges, have become a cornerstone of modern programming education. Conventional autograders are constrained to a key limitation: they are designed to evaluate only "correctness", the binary pass/fail of code tests. By focusing only on this metric, a significant amount of the pedagogical value can be diminished. This correctness-focused approach represents an important pedagogical drawback in programming education. It provides a limited and potentially misleading view of students' real understanding, as a "correct" answer can be achieved using "wrong", or even "deceptive", steps. The core of the problem is: traditional paper assessments are slow and inefficient, while assessments automated by autograders struggle to verify the student's reasoning.

This article introduces a competency-driven framework for designing autograder test suites, that moves beyond the correctness of a problem, and focuses on enabling a more meaningful evaluation scheme. This approach lies in deconstructing a programming problem into multiple competencies that should be tested, such as the use of conditional or loop structures, or correctly dealing with edge cases. Based on this, the framework guides the creation of a dedicated test suite for each problem, intentionally designing and weighting each test case to best represent the evaluation process. This reframes the autograder from a simple "yes/no" checker into a comprehensive diagnostic tool.

To validate this framework, an extension for the Machine Teaching (MT) platform, an online judge system used at the Universidade Federal do Rio de Janeiro, was developed. This implementation focuses on allowing synchronous assessments to be taken within this system. We use this tool to investigate if our approach can enhance the pedagogical value of automated assessment, without sacrificing its efficiency. The goal of this work is to contribute to a structured and replicable framework for educators to build more informative automated evaluations, aiming to harmonize the scalability of autograders and the depth of human-led assessment processes.

This article describes the framework, the experimental tool and the validation of our method. Section 2 provides the theoretical foundation of our framework, while Section 3 explains the methodology, which includes the research phases and the experimental design. Section 4 presents the proposed competency-driven framework. Finally, the results are discussed in Section 5, and the conclusions are drawn in 6.

2. Background and Related Work

This section reviews the literature on automated assessments in programming education. Firstly, we discuss the state of the art and its pedagogical limitations. Subsequently, we present the theoretical and technical foundations for a solid competency-based approach.

2.1. Programming autograders state of the art and its limitations

Automated assessment tools have become essential in modern programming education. For instructors, they significantly reduce grading workload—by about 35% on average [Langove and Khan 2024]—while for students, immediate feedback increases engagement, raising assignment submission rates from 70% to 88%. Despite these advantages, most conventional autograders rely on the assumption that producing the correct answer is sufficient. In practice, they evaluate only one dimension: functional correctness. While widely used tools like the Virtual Programming Lab (VPL) allow custom test case definitions, systematic reviews show that most autograders still provide binary feedback without assessing the underlying methodology [Messer et al. 2024]. This creates a gap between automated grading and comprehensive evaluation. Instructors report that they must still manually verify aspects such as the correct use of structures like conditionals or loops [Barbosa et al. 2023]. Thus, online judges alone are often insufficient for end-to-end assessment, requiring complementary mechanisms to evaluate solution quality. These perspectives correspond to two distinct notions of correctness: functional correctness, which verifies whether the program produces the expected output, and methodological correctness, which evaluates whether the student applied the appropriate programming concepts.

This binary-correctness approach is limited and can lead to an inaccurate evaluation of student knowledge. When researchers compared challenge-based assessments, in which students have to solve a small number of problems that test multiple concepts simultaneously, with competency-based assessments, in which concepts are decoupled into small focused problems, it revealed that the first method is far less accurate when assessing student learning progress [Sidhu et al. 2021]. The challenge-based approach resulted in a final exam failure rate of 42%, as a misunderstanding of a single concept could prevent a student from completing the entire problem. By switching to the competency-based model, the failure rate decreased to just 6%, indicating that a granular evaluation provides

a more fair and accurate measure of a student's actual knowledge. This sense of unfairness is noticed by students and may contribute to frustration.

2.2. Competency-Driven Learning Approach

An alternative to these limitations is a pedagogical shift from task-based assessment to competency-based assessment. A competency is defined as an individual's proficiency in applying knowledge, skills, and dispositions to perform tasks in a given context [Cruz et al. 2025]. In programming education, this includes abilities such as implementing loops, selecting appropriate data structures, or handling edge cases. This perspective aligns with the distinction between "knowledge assessment" and "skill assessment" [Hettiarachchi et al. 2013]. Knowledge assessment is typically convergent, aiming at a single correct answer through structured formats such as multiple-choice or short-answer questions, which are easy to grade but evaluate a limited set of skills [Crisp 2009, McAlpine 2002]. Skill assessment, in contrast, is divergent and focuses on open-ended or authentic tasks that provide richer evidence of higher-order cognitive abilities, although they are more demanding to design and evaluate [Crisp 2009, McAlpine 2002, Hettiarachchi et al. 2013]. Adopting a competency-based perspective allows automated assessment to move beyond purely convergent evaluation. However, Competency-Based Education (CBE) remains underexplored in programming autograding, revealing an important research gap.

Assessing methodological competencies requires analyzing not only program output but also the code itself. The literature suggests combining static and dynamic techniques in a hybrid analysis approach [Arifi et al. 2015]. Static analysis inspects source code structure without execution, while dynamic analysis evaluates program behavior during runtime. According to a five-level evaluation scale [Combéfis 2022], most current autograders assess only code semantics—whether the program passes test cases—corresponding to functional correctness. Evaluating code quality requires deeper analysis. Our framework adopts this hybrid perspective by integrating dynamic analysis for functional competencies and static analysis for methodological ones. Bibliometric studies identify "static analysis" and "test generation" as prominent topics in automated assessment research [Paiva et al. 2023].

Previous approaches, such as the "A-Learn EvId" method [Porfirio et al. 2021], detect programming skills through detailed Abstract Syntax Tree analysis. In contrast, our work emphasizes the intentional design of test cases aligned with learning objectives. By defining the competencies to be assessed beforehand, instructors can create diagnostic test suites in which each test case targets a specific competency. This structured automation also mitigates inconsistencies inherent in manual grading, which has been shown to present low reliability, with Krippendorff's α values as low as 0.2 for correctness [Messer et al. 2025]. The proposed framework therefore aims to provide an evaluation that is efficient, consistent, and more objective than manual grading while offering deeper insight than simple pass–fail analysis.

3. Methodology

This research adopted a constructive approach structured in three phases to address limitations of traditional autograders identified in the literature. First, we defined the functional

and pedagogical requirements of a competency-based assessment system, emphasizing the isolation of specific skills through granular testing. Second, we designed the data model and implemented the framework within the Machine Teaching Platform, enabling weighted test cases and hybrid analysis. Third, we conducted an experimental study comparing the framework with traditional assessment in terms of grading efficiency and reliability.

To empirically validate the framework, it was implemented as an extension of the Machine Teaching (MT) platform, an established Learning Management System used in programming courses at the Universidade Federal do Rio de Janeiro (UFRJ) [Folloni Guarilha 2025, Moraes et al. 2021].

3.1. Tool Design and Data Modelling

The implementation required a data model capable of representing the framework concepts. This involved extending the existing "problem" structure and introducing entities to manage relationships between exams, sessions, and submissions. The resulting schema translates pedagogical analysis into the following structures:

Topic-related Questions: Each exam, represented by the `Evaluation` table, stores metadata such as title and availability dates. Exams are composed of theoretical or practical questions selected or created by the instructor to assess specific topics.

Weighted Questions: The `EvaluationProblem` table links problems to a given exam and defines their weight (point value), enabling differentiated importance among questions and the competencies they represent.

Weighted Test Cases: The `EvaluationProblemTestCase` table associates specific test cases with each problem and assigns them individual weights. This enables fine-grained evaluation, allowing critical cases (e.g., edge conditions) to contribute more to the final score than basic functionality tests.

Beyond the data structure, the framework's implementation required specific functionalities to translate pedagogical competencies into an automated assessment.

Support for Hybrid Analysis: Our framework requires assessing competencies based on a hybrid analysis, with both dynamic and static testing. The platform's autograding system was built to support this approach:

- **Dynamic Analysis:** The classic method of running code with inputs and checking outputs, ideal for assessing functional competencies.
- **Static Analysis:** The instructors' ability to inspect the source code after the initial grading, which is used for assessing methodological competencies.

Test Case Configuration: The user interface allows instructors to configure each test case individually. This is where they map a specific test to a competency, assign its weight to reflect its pedagogical importance, and set its visibility as public (for formative feedback during the exam) or private (for summative grading).

3.2. Experimental Setup

The study uses a within-subjects design in which the same group of students completes two assessments covering isomorphic content (the semester's second major exam, *Prova 2*). Students were informed that their final grade would correspond to the highest score obtained across both attempts to encourage participation in both conditions.

1. **Experimental Condition (MT Platform):** Conducted on November 19th, 2025, in the university lab using the MT platform with the proposed competency-driven framework.
2. **Control Condition (Traditional):** Conducted on November 26th, 2025, as a traditional pen-and-paper exam graded manually by the instructor.

Both exams were designed by the same instructor and are isomorphic, requiring the same programming competencies while using different narratives to avoid memorization. Although this fixed chronological order poses a potential learning effect threat, the isomorphic design of the problems was specifically chosen to mitigate this factor

3.3. Data Collection and Metrics

The experiment analyzed the final cohort of $N = 38$ students who completed both assessments. To evaluate the framework, we defined quantitative and qualitative metrics collected during the study.

A key hypothesis is that the framework shifts instructor effort from grading to assessment design. To measure this, we recorded two **instructor efficiency metrics**:

- **Creation Time:** Time required to design the assessment, including problem preparation, answer key, and test case configuration.
- **Grading Time:** Total time required to correct submissions. For the control condition, this corresponds to manual grading time; for the experimental condition, it includes the completion of both grading stages on the platform.

To evaluate the pedagogical impact, a survey was administered after the assessments and before grade release, measuring the following **student perception metrics**:

- **Comfort:** Student comfort when coding with IDE support compared to writing on paper.
- **Difficulty:** Perceived difficulty of each assessment format.
- **Confidence:** Students' perceived performance in each condition.

4. The Competency-Driven Framework

This section details the proposed competency-driven framework, providing a structured methodology for designing autograder test suites. The framework's primary objective is to bridge the gap between the efficiency of automated assessment and the pedagogical depth of manual evaluation. It aims to formalize a process for instructors to build assessments that shift the focus from a singular, monolithic problem to a granular evaluation of its underlying competencies.

This framework deconstructs competencies into two categories, which directly map the stages of the grading process:

1. **Functional Competencies:** Skills that could be reliably verified by I/O testing (dynamic analysis), such as the problem's main goal, addressing edge cases, and ensuring that solutions are not hardcoded.
2. **Methodological Competencies:** Skills that require human inspection to assess (static analysis). This includes "correctness of methodology" (use of recursion vs. iteration, not using a specific library), as well as aspects of readability and maintainability.

To implement this framework, the assessment lifecycle follows three stages:

Definition of Competencies: The instructor selects problems aligned with the learning objectives and decomposes them into specific functional and methodological competencies, rather than treating the task as a single unit.

Configuration of Weights and Tests: The identified competencies are mapped to test cases (dynamic analysis) and constraints (static analysis). Weights are assigned to individual test cases according to the taxonomy in Table 1, prioritizing deeper reasoning over superficial correctness.

Hybrid Grading: Grading is performed in two phases. The automated system first evaluates functional competencies and produces a preliminary score. The instructor then performs a focused manual review addressing the remaining methodological constraints.

4.1. Competency-Driven Test Suite

The first stage of the framework provides the mechanism for assessing "Correctness of Functionality". The core of this stage is the intentional design of test cases. The process requires the instructor to first define the functional competencies to be assessed. These attributes could include the ability to:

- handle general/normal cases.
- correctly handle edge cases, like zero, invalid inputs or empty strings.
- provide a robust, non-hardcoded solution (a competency that can be inferred by testing non-obvious or large inputs).

The instructor must then design specific test cases to cover each one of these. Each functional competency should be mapped into a way to design the test cases, following a pedagogical goal. The process of translating some pedagogical goals into test cases is shown in Table 1.

Table 1. Mapping Functional Competencies to Test Case Design

Functional Competency	**Test Case Design**	**Pedagogical Goal**
Handles General Cases	Create tests with standard, expected inputs ('factorial(5)').	Assesses the core logic of the algorithm.
Handles Logical Edge Cases	Create tests for boundary conditions ('factorial(0)').	Assesses understanding of the base case.
Handles Empty Inputs	Create tests for empty structures ('count_vowels("")').	Assesses loop robustness and prevents null errors.
Handles Invalid Inputs	Create tests for inputs outside constraints ('factorial(-1)').	Assesses defensive coding and error handling.
Handles Hardcoding	Create a non-obvious, large-input test ('factorial(42)').	Assesses scalability, not just 'if/else's.
Handles Algorithmic Efficiency	Create a large dataset that will time out a brute-force solution.	Assesses use of the correct algorithmic structure.
Handles Non-Trivial Cases	Create tests that are designed to break naive assumptions.	Assesses the robustness of the student's logic.

The implementation of our system enables this framework by allowing an instructor to assign a specific numeric weight to each test case. This weight reflects the pedagogical importance of the test. This weighting mechanism enables instructors to implement a formal, pedagogically-justified grading strategy, moving beyond simple test case counts. For instance, an instructor can use the table to justify a "60/40" split for a factorial problem. In this scheme, 60% of the grade could be assigned to the "Handles General Cases" tests, while 40% is assigned to the "Handles Edge Cases", "Handles Empty Inputs" and "Handles Invalid Inputs" tests. This disproportionate weighting is justified, as these tests are highly informative, assessing a deeper level of student reasoning. The result of this stage is, therefore, not a simple pass/fail verdict but a granular diagnostic score. This automated grade approximates the evaluation a human instructor would provide, which is known to be time-consuming. It provides a more accurate and fair assessment of a student's functional knowledge, while setting a solid foundation for the later manual review.

4.2. Residual Methodological Verification

The second stage of the framework begins after the automated grading is complete, with the human review serving primarily as a verification step. The platform presents the diagnostic score from Stage 1 to the instructor, with the student's source code. This allows instructors to quickly address their code reviews. Their role is transformed from a time-consuming evaluation, from-scratch, into a fast and focused review, to assess the pre-defined methodological competencies. This is where they perform the "static test" to check for things the dynamic tests cannot, such as:

- Adherence to constraints: "Did the student import math.factorial?".
- Use of required structures: "Was recursion or a for loop used as specified?".
- General Readability or Maintainability: "Are variable names meaningful?".

This allows the instructor to apply their final pedagogical judgment and adjust the grade in seconds, rather than minutes.

4.3. Value and Limitations

This hybrid framework provides two distinct benefits for educators. First, it accelerates existing workflows: instructors can apply the framework to their current assignments and save hours of grading time by automating the laborious workload of grading assessments. Second, it empowers new, better assessments: instructors are no longer limited to designing problems that are "autograder-friendly". They can now create more pedagogically-rich problems, that the two-stage framework can fully assess, without being restricted in that sense. In both cases, professors win time, and students win fairness and more granular feedback.

It is important to define the scope of this framework. It does not fully automate the assessment of all competencies, such as Readability or complex "correctness of methodology". Instead, it formalizes the process of these checks as a key part of the Stage 2 manual review. By automating the functional component, the framework makes this "static test" faster and systematic. A clear direction for future work is the integration of AI or LLM-based tools to begin automating aspects of this second, manual stage.

While this study utilizes the Machine Teaching platform, the proposed framework is platform-agnostic. It requires any automated assessment system that supports two core

features: (1) Weighted Test Cases, allowing instructors to assign partial credit to specific tests rather than binary pass/fail; and (2) Configurable Visibility, enabling distinctions between 'public' tests (formative feedback) and 'hidden' tests (summative assessment). Systems like Moodle (with VPL) can adapt this framework, provided that these configurations are available.

5. Results

This section presents the quantitative data regarding instructor efficiency and the comparative analysis of student performance.

5.1. Instructor Efficiency Analysis

The most significant finding observed in this workflow was the shift in instructor workload distribution. We compared the time investment required for the traditional paper-based assessment against the proposed framework. As hypothesized, the framework required a higher initial investment on **exam creation**. Designing the competency-driven exam on MT required **200 minutes**, involving the mobilization of creativity for problem narratives and the intentional design of weighted test cases, with all data insertions into the system. In contrast, the traditional paper exam required approximately **120 minutes** to design, as questions were intentionally "simplified" to facilitate manual grading.

The efficiency gain (return on investment) in the **grading** phase was substantial, confirming the effectiveness of this framework and this two-stage hybrid approach as well.

- **Control Baseline:** For the paper assessment, the instructor recorded a total grading time of **8 hours (480 minutes)**This averages to approximately **12.3 minutes per student**.
- **Experimental Result:** For the MT exam, the total time for the complete grading process was recorded at approximately **17 minutes**. Because the autograder had already filtered functional correctness, the instructor's role was reduced to a rapid verification of methodological constraints. This averages the grading time per student to less than 25 seconds, confirming the validity of this approach.

This result represents a grading workload reduction of approximately **97%**. When considering the total cycle time (Creation + Grading), the proposed framework reduced the total instructor workload from 10 hours (Baseline) to 3.6 hours (Experimental), a **64% net gain** in efficiency, despite the increased complexity of exam creation.

5.2. Student Performance Analysis

To validate the framework's assessment reliability, we compared the performance of $N = 38$ students who completed both the MT and the traditional Paper exam. We examined whether the automated approach introduced significant grade distortions compared to the manual baseline.

Table 2. Comparison of Grade Distribution (MT vs. Paper)

Metric	Experimental (MT)	Control (Paper)
Mean (μ)	6.87	7.08
Median	8.60	7.10
Std. Deviation (σ)	3.68	2.64
Pass Rate (≥ 7.0)	60.5%	56.4%

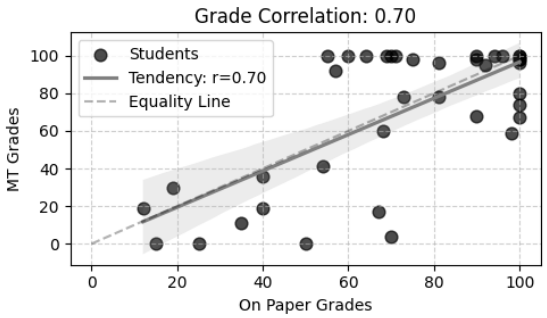

Figure 1. Comparison of Grade Distribution (MT vs. Paper)

As shown in Table 5.2, the **Mean (μ)** scores were remarkably similar, indicating that the framework maintained the difficulty level of the control evaluation. However, the MT group exhibited a higher **Median** and higher volatility. This suggests that while immediate feedback (public test cases) helped competent students refine their code, the binary nature of automated compilation penalized errors more strictly than manual partial credit. Despite these structural differences, the strong **Pearson correlation** of $r = 0.70$ ($p < 0.001$), as shown in Figure 1, provides strong evidence of concurrent validity with traditional grading, though deeper psychometric analysis is required to confirm individual competency mappings.

5.3. Student Perception

To evaluate the student experience, a survey was conducted with the $N = 31$ participants who completed both assessments. Results show a strong **preference** for the competency-driven automated format: 83.9% of students would choose the MT platform for future exams. This preference is consistent with reported **confidence** levels, with 61.3% feeling more confident coding in the IDE and only 19.3% preferring paper.

A key feature of the framework is the use of public test cases as formative feedback. The data strongly supports this mechanism: 93.6% of students reported that test cases helped them identify errors that would likely go unnoticed in paper exams.

Perceptions of difficulty were more balanced. While 45.2% considered the paper exam more challenging, 29.0% rated the automated assessment as harder and 25.8% perceived both as equivalent. This suggests that the automated environment does not necessarily simplify the assessment but changes its nature. Although the IDE provides support, the strictness of compilation enforces precise functional correctness, reducing the ambiguity often present in manual grading.

Qualitative responses reinforced these findings. Several students reported that coding in an IDE felt more natural than relying on a "mental compiler" during paper

exams. They also noted that automated tests exposed logical flaws they might otherwise overlook, allowing them to correct errors before submission.

5.4. Discussion

The results validate the proposed framework as a scalable alternative to traditional assessment. The strong correlation ($r = 0.70$) indicates that this approach measures competencies comparable to those assessed in manual exams, addressing concerns that autograders evaluate only superficial aspects of code. The findings support a shift in the instructor's role from "marker" to "designer". By investing time to design comprehensive test suites (200 minutes), the instructor achieved a 97% reduction in grading time, transferring the pedagogical effort from reactive correction to proactive definition of assessment criteria.

The grade distribution reveals a "Compiler Effect". The higher median in the experimental group (8.6 vs 7.1) suggests that immediate feedback allows students to iteratively correct mistakes before submission. However, this should not be interpreted as reduced rigor: in both formats students must design working solutions, and improvement still depends on their ability to produce better code. The automated environment implements an instructor-defined feedback loop through visible test cases. By running these tests before final submission, students refine their solutions and reasoning, turning the test suite into a pedagogical scaffold that supports competency development.

In contrast, paper-based exams impose additional cognitive load, as students must recall syntax and formatting typically supported by IDEs. This often leads instructors to simplify problems and adopt highly structured questions to manage grading time. Automated compilation also introduces stricter evaluation: unlike human graders who may award partial credit for broken logic, the system enforces functional correctness. Rather than replicating human grading, this framework encourages iterative improvement.

6. Conclusion

Conventional autograders typically restrict evaluation to functional correctness, often failing to assess the underlying reasoning or adherence to methodological constraints. This work addressed this gap by proposing a **"Competency-Driven Framework"** that breaks the assessment into two stages: an automated validation of functional competencies followed by a focused, human-in-the-loop review of methodological skills.

The efficiency gains were substantial. By shifting the pedagogical investment to the exam creation phase, the framework reduced the grading workload by 97% and the total assessment cycle time by 64%. This efficiency releases the instructor from the mechanical burden of repetitive code analysis and marking, allowing for a reallocation of effort toward higher-value activities, such as personalized feedback. This efficiency did not come at the cost of validity. The strong correlation ($r = 0.70$, $p < 0.001$) between the experimental automated exam and the traditional paper control confirms that this framework assesses the same underlying programming competencies.

In conclusion, this work demonstrates that automated assessment, when supported by a rigorous design framework, is not merely an efficiency improvement, but a pedagogical evolution. It transforms the instructor's role from a reactive evaluator to a proactive "assessment architect", ensuring that scalability and educational quality can coexist in the expanding landscape of Computer Science education.

Use of Generative AI

Generative AI technologies (specifically Gemini and ChatGPT) were used in the preparation of this work for two main purposes: (1) to assist in grammatical review and the refinement of technical writing style; and (2) to support the exploratory analysis and interpretation of the quantitative metrics derived from the experiment. The authors reviewed all AI-generated suggestions and take full responsibility for the content and validity of the final text.

References

Arifi, S. M., Abdellah, I. N., Zahi, A., and Benabbou, R. (2015). Automatic program assessment using static and dynamic analysis. In *2015 Third World Conference on Complex Systems (WCCS)*, pages 1–6.

Barbosa, A. d. A., Costa, E. d. B., and Brito, P. H. (2023). Juízes online são suficientes ou precisamos de um var? In *Anais do III Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, pages 386–394, Porto Alegre. Sociedade Brasileira de Computação.

Combéfis, S. (2022). Automated code assessment for education: Review, classification and perspectives on techniques and tools. *Software*, 1:3–30.

Crisp, G. (2009). *Assessment in Higher Education: Professional Practice and Future Challenges*. Routledge.

Cruz, L. S., Santos, J. A. M., Coutinho, L. d. A. H., and Salvador, L. N. (2025). A reference model for presentation of studies in competency-based education in engineering and computing. In *Anais do Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, pages 59–71, Porto Alegre. Sociedade Brasileira de Computação.

Folloni Guarilha, H. (2025). Implementation of synchronous assessments on the machine teaching platform. Universidade Federal do Rio de Janeiro, Trabalho de Conclusão de Curso.

Hettiarachchi, E., Huertas, M. A., and Mor, E. (2013). Skill and knowledge e-assessment: A review of the state of the art. *IN3 Working Paper Series*, (2013):1–22.

Langove, S. A. and Khan, A. (2024). Automated grading and feedback systems: Reducing teacher workload and improving student performance. *Journal of Asian Development Studies*, 13(4):202–212.

McAlpine, M. (2002). *Principles of Assessment*. CETIS. Available online.

Messer, M., Brown, N. C. C., Kölling, M., and Shi, M. (2024). Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computing Education*, 24(10):1–43.

Messer, M., Brown, N. C. C., Kölling, M., and Shi, M. (2025). How consistent are humans when grading programming assignments? *ACM Transactions on Computing Education*, 25(4).

Moraes, L. O., Pedreira, C. E., Delgado, C. A. D. M., and Freire, J. P. (2021). Supporting Decisions Using Educational Data Analysis. In *Anais Estendidos do Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia)*, pages 99–102. SBC.

Paiva, J. C., Figueira, A., and Leal, J. P. (2023). Bibliometric analysis of automated assessment in programming education: A deeper insight into feedback. *Electronics*, 12:2254.

Porfirio, A., Pereira, R., and Eleandro, M. (2021). A-learn evid: A method for identifying evidence of computer programming skills through automatic source code assessment. *Revista Brasileira de Informática na Computação (RBIE)*.

Sidhu, G., Srinivasan, S., and Muhammad, N. (2021). Challenge-based and competency-based assessments in an undergraduate programming course. *International Journal of Emerging Technologies in Learning (iJET)*, 16(23):17–28.