POLITECNICO DI TORINO
Repository ISTITUZIONALE

Investigating Web Project Assessment in an AI World

Original
Investigating Web Project Assessment in an AI World / Russo, F., Saenz, J.P., De Russis, L.. - (2026), pp. 1-5. (CHI '26: CHI Conference on Human Factors in Computing Systems Barcelona (ESP) April 13–17, 2026) [10.1145/3772363.3798887].

Availability:
This version is available at: 11583/3008077 since: 2026-04-24T12:14:09Z

Publisher:
ACM

Published
DOI:10.1145/3772363.3798887

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

06 October 2026

Investigating Web Project Assessment in an AI World

Francesca Russo
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino, Italy
francesca.russo@polito.it

Juan Pablo Saenz
Politecnico di Torino
Turin, Italy
juan.saenz@polito.it

Luigi De Russis
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino, Italy
luigi.derussis@polito.it

Abstract

Project-based assessment in university web application courses is widely used because it shows how students integrate skills and knowledge across development. However, in large cohorts, project assessment becomes time-intensive and it is difficult to keep evaluation consistent across students. Automated support can reduce workload, but may limit instructors' agency during the assessment. To understand instructors' challenges and needs, we conducted semi-structured interviews followed by a sketching exercise with seven university instructors who regularly assess web application projects as part of the final exam. By analyzing data from both activities, we derived four themes and related implications. Key themes include critical thinking as the boundary for acceptable GenAI use and instructors' preference for instructor-in-the-loop tools that automate pre-checks and extract evidence to drive oral questioning. These findings can inform the development of systems that can support instructors in assessing web application projects at scale while keeping them in control of decisions.

CCS Concepts

• **Human-centered computing** → **User studies**; • **Applied computing** → *Education*.

Keywords

instructors, assessment, project assessment, interview

ACM Reference Format:
Francesca Russo, Juan Pablo Saenz, and Luigi De Russis. 2026. Investigating Web Project Assessment in an AI World. In *Extended Abstracts of the 2026 CHI Conference on Human Factors in Computing Systems (CHI EA '26), April 13–17, 2026, Barcelona, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3772363.3798887

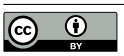
1 Introduction and Background

Assessing students' work is a central part of teaching. It helps instructors verify what students have understood while supporting students' learning by providing feedback. Nonetheless, assessment should not only test whether students can memorize information but it should also evaluate higher-order skills, such as critical thinking, creativity, and problem solving. In university computer science courses, being able to transfer knowledge into practice when facing complex problems that mirror real-world contexts is crucial [8, 12], as such context require building artifacts, making design decisions, revising work, and justifying trade-offs. To better reflect these realities, instructors often rely on take-home assessments [12], in which students develop a large project and integrate knowledge with skills such as planning, designing, implementing, and debugging. However, take-home assessment faces new challenges due to the adoption of Generative Artificial Intelligence (GenAI) [3, 11, 14]. In those cases, instructors increasingly report concerns that students may use GenAI to generate code without developing the underlying understanding, making it harder to assess students' knowledge and authorship from only the submitted project.

To address this concern, instructors add an additional layer that focuses on verification of understanding and authorship, combining a take-home project with an oral examination [10, 14]. Oral exams can support more valid assessment by probing students' reasoning, discussing key design choices, and clarifying the extent to which students can justify and extend their submitted work. At the same time, oral discussions can provide both summative and formative value, enabling instructors to assign grades while also giving feedback that students can use to improve [4, 9].

However, despite such benefits, evaluating high-complexity software projects in university courses poses significant challenges for instructors [2]. The development of individual applications based on functional and non-functional requirements, with the freedom to design the architecture, component integration, and user interface, combined with a high number of students and limited evaluation time, can take a considerable amount of time and effort.

Prior literature documents a range of tools for supporting grading workflows in computer science education [13, 15]. Some approaches assess code quality without running the program, inspecting syntax to surface possible misconceptions and to check for the presence or absence of particular constructs [18]. Other approaches execute programs and run unit tests to verify if the outputs match expected results, but they provide limited insight into code quality beyond functional correctness [1]. For web projects, Siochi et al. presented WebWolf [17] which allows instructors to write JUnit-like tests for simulating user actions and assert browser state by driving a headless browser via Selenium, while Peveler et al. [16] presented a system that runs students' project in isolated Docker containers and supports manual grading. More recently, Large Language Models (LLMs) have been explored as rubric-aware assistants that draft grades [7, 19]. Building on this line of work, Chen et al. [6] conducted a formative study on grading project reports and developed CoGrader, a human–LLM collaborative assessment assistant.



This work is licensed under a Creative Commons Attribution 4.0 International License.
CHI EA '26, Barcelona, Spain
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2281-3/26/04
https://doi.org/10.1145/3772363.3798887

Nevertheless, to our best, there is no research that investigates instructors' practices and needs in assessing complex web application projects[1]. To fill this gap, we conducted a formative study with seven instructors who assess web application projects through a final take-home project followed by an oral exam. The study combines semi-structured interviews with a sketching exercise. In the interview, we identified the challenges they encounter; in the sketching exercise, instead, we explored what kinds of system they envision to support the assessment of these exams. Taken together, our results highlight a workflow where documentation guides the initial pass; code inspection, and evidence extraction drive probing questions, and the oral discussion supports judgments about understanding and authorship. Finally, instructors prefer tools that speed up verification and evidence gathering without taking over assessment decisions.

2 Formative Study

We conducted a study with three main instructors and four collaborators from four different web applications courses. In Appendix A, we report the full set of initial questions. The study was structured in two parts:

(1) **Semi-structured interview**: in this first phase, our goal was to characterize instructors' current grading practices and understand what influences decision-making. We started with contextualization questions to assess each participant role in the course and the years of experience. Then, we asked participants to show a representative web project and to walk us through the workflow starting from the delivery up to giving the mark to the student. During this walkthrough, we investigated how grades are determined in practice, including: *"How is the grade composed?"*, *"Is there a rubric to follow?"* and *"When designing the rubric, how do you decide the maximum point value for each criterion?"*.
(2) **Sketching exercise**: this phase was introduced by a set of open questions. We asked participants about the challenging aspects of oral exams based on students' projects and whether they already use any automation tool to support the assessment. These questions were included to contextualize their current practices and highlight pain points. Then, we asked participants to reflect on how they could be supported in improving the project grading phase. They were allowed to come out with whatever solution they could think of. Finally, we asked them to translate their envisioned solutions into a sketch. Participants were free to draw and, when they felt uncertain, the interviewer summarized relevant parts of the interview to prompt them to externalize those elements in the sketch. There was no time limit for completing the sketch.

3 Results

The study was conducted in-person with 4 participants, while the remaining 3 were online during October and November 2025. They were semi-structured interviews in which, upon the participants' responses, we asked them to deepen their answers. Table 1 summarizes key information about the participants and courses they teach. On average, each session lasted 68.14 m (min=33 m , max=103 m, SD=27.56) and all the interviews were in Italian. The interviews were audio recorded (with the prior consent of the participants) and later transcribed.

[1]In this article, by "complex web application", we mean a multi-page application with authentication and a client–server architecture, where the client and server communicate via APIs and the server stores data in a relational database.

3.1 Courses Topics and Grading Workflow

The 4 courses are offered within a university-level Computer Engineering program, and each includes a take-home project at the end of the course, followed by an oral discussion for grading. "Introduction to web application" is a Bachelor's Degree course that covers foundational topics for designing and developing web applications. It introduces basic concepts of web design and the architecture of web applications, including an overview of the HTTP protocol, the HTML5 language and CSS3, with particular attention to responsive design. The course also covers client-side development with JavaScript and server-side development in Python. Data persistence is covered through relational databases. Finally, the course introduces a range of development tools used both inside and outside the browser.

The remaining courses are Master's Degree courses that cover core concepts and practices for building modern web applications. They start with notions of web design and the advanced aspects of JavaScript. On the front end, the course focuses on React while, on the back end, it adopts Express as web servers, data exchange using JSON and HTTP APIs, and data persistence through a SQLite database. Finally, it covers security-related topics, including authentication.

Across courses, the take-home exam follows a common structure: instructors publish the project text, students develop the project over a fixed time window, and grading is finalized through an in-person oral discussion based on project inspection and questioning.

Publication of the text. The instructor publishes an exam text that includes (i) logistics and rules, (ii) technical constraints (required stack), and (iii) functional and non-functional requirements written in natural language. After the publication of the text, students have about twenty days to complete their project.

Submission intake and instructors preparation. Students submit the project artifact online. Before oral sessions, instructors prepare a grading grid as rubric, shared with collaborators in the course. The rubric is composed of macro-voices that reflect functional and non-functional requirements.

In person oral examination. The exam lasts approximately 30 minutes. It focuses on project execution and inspection to verify the correctness, plus discussion to verify authorship and key design choices.

3.2 Thematic Analysis

We conducted an inductive thematic analysis [5] of the study outcomes. Themes are presented below, followed by a brief discussion of their implications.

3.2.1 Documentation as gatekeeper of the project. Across interviews, instructors identify the README (i.e., the documentation of the project produced by students) as the **entry point** of the oral assessment. Several described it as the very first artifact they look

Participant	Age	Years Teaching	Role	Course	Course Language	Students Per Year
P1	55-60	7	Main instructor	Web Applications 1	English	~ 280
P2	30-35	6	Main instructor	Introduction to Web Applications	English	~ 150
P3	30-35	5	Collaborator	Applicazioni Web 1	Italian	~ 250
P4	55-60	3	Main instructor	Web Applications	English	~ 200
P5	30-35	3	Collaborator	Applicazioni Web 1	Italian	~ 250
P6	25-30	1	Collaborator	Applicazioni Web 1	Italian	~ 250
P7	50-55	3	Collaborator	Web Applications	English	~ 200

Table 1: Participant overview, including age range, years of teaching experience for the related course, role in assessment (main instructor or collaborator), the course assessed, the course language, and average number of enrolled students per year.

for and read: P5 noted that "*the first thing I look for is the README*," while P1 similarly said "*I start by reading the documentation, while the application is starting up*". P4 highlights this ordering: "*the first thing I do is read the README*," using it to "*get an idea of the application*" before moving to code inspection. Likewise, P6 framed it as a correctness and completeness check: "*First of all, I look at the project's README to check that it has been filled out correctly*," ensuring that required parts are present.

Instructors also used the README to identify where to focus during the discussion. In particular, they treated API documentation as a trail of students' design reasoning. P1 explained that they "*pay particular attention to APIs because they're what let me understand more clearly how they reasoned: how responsibilities are split between client and server, what data is exchanged back and forth, and therefore, you can immediately infer how the database is structured, how the server is designed, what functions exist, and so on, and how they reasoned about where the application state lives*". This then becomes the basis for targeted questions (e.g., "*why is this API like this?*"). P4 likewise described the API section as the part they "*look at very closely to understand how the client and server communicate*," adding that when there is something "*I don't understand, I have them explain it to me*."

Finally, while the README offers a comprehensive overview of the project structure, instructors still need to inspect the code to verify that what is described in the README is actually implemented in the project. As P4 explained, they check that "*something they wrote in the README corresponds to what they have actually done*".

3.2.2 Blind spots and cross-grader misalignment. Throughout the interviews, instructors described assessment as inherently **sampling-based**: under strict time constraints, they can only check a subset of behaviors and implementation details, which increases the likelihood of blind spots and uneven penalties. P2 highlighted how workload and fatigue can reduce the rigor of later evaluations: *"Tiredness, [...] means that you might not look at it with the same care as you did the first one."*. P6 similarly stressed that *"It's not actually possible to check everything without reading code line-by-line"*, and that *"it becomes a bit random. Sometimes I penalize one student for an error, but I can miss the same error in another student's work, so they aren't penalized"*. In other words, non-blocking issues, corner cases, and hidden implementation mistakes can slip through simply because there is not enough time to exhaustively verify each project.

Instructors noted that maintaining consistent grading across students is difficult to guarantee when **multiple instructors** are involved. P5 explained that when multiple people evaluate projects, alignment conversations help but cannot eliminate differences: "*Solving every misalignment is impossible*". P3 described repeatedly checking ambiguous cases with the lead instructor so that *"We don't use double standards"*. Nonetheless, even with a shared rubric, assigning partial points can remain subjective: P1 emphasized that cross-cutting issues make it hard to attribute problems cleanly, concluding that *"The evaluation is always subjective"*. Finally, instructors noted that misalignment is not only about points, but also about what content to prioritize during questioning: P7 argued it matters to know *"What to give importance to, and what not to"*, explaining that *"it's one thing for it to appear on a slide; it's another if you just skim over it quickly versus spending fifteen minutes discussing it during lectures—giving examples, showing pros and cons [...], so it makes sense to ask them a question during the oral exam otherwise, no"*.

3.2.3 Critical Thinking as the Boundary for GenAI. Instructors described GenAI as increasingly embedded in students' workflows, and positioned this as reflecting real-world practice, that is why it is not something to prohibit. As P1 explains, "*It seems right to me that, during the exam, a student should have the same tools that they'll have in the working world*". On the other hand, using GenAI is not very different from writing the code with a collegue: as underlined by P1 the students can "*copy it from a collegue, [...] but then, during the oral exam, if I ask what a line of code does, they must be able to explain it, even if it is AI-generated*". However, this acceptance came with a clear boundary: students should not be replaced by AI. Indeed, P1 continues emphasizing that the core problem is not AI-use per se, but the absence of critical engagement: "*I don't see anything wrong with them using AI; the problem is when critical thinking is missing*". In practice, instructors rely on asking questions during the oral exam whenever something "*doesn't add up*", for instance, when code diverges from what has been showed during the course or introduces unfamiliar constructs, "*because it is written by ChatGPT and similar*" (P4). This pushes assessment towards the explanation and the rationale behind the code. For example, P3 described asking targeted "why" questions "*to check that it wasn't just copy-pasted code*" and to test understanding of what a snippet does.

3.2.4 Automation is welcomed with the instructor in the loop. When asked to reason about how they could be supported in improving the phase of grading projects, instructors consistently pointed to

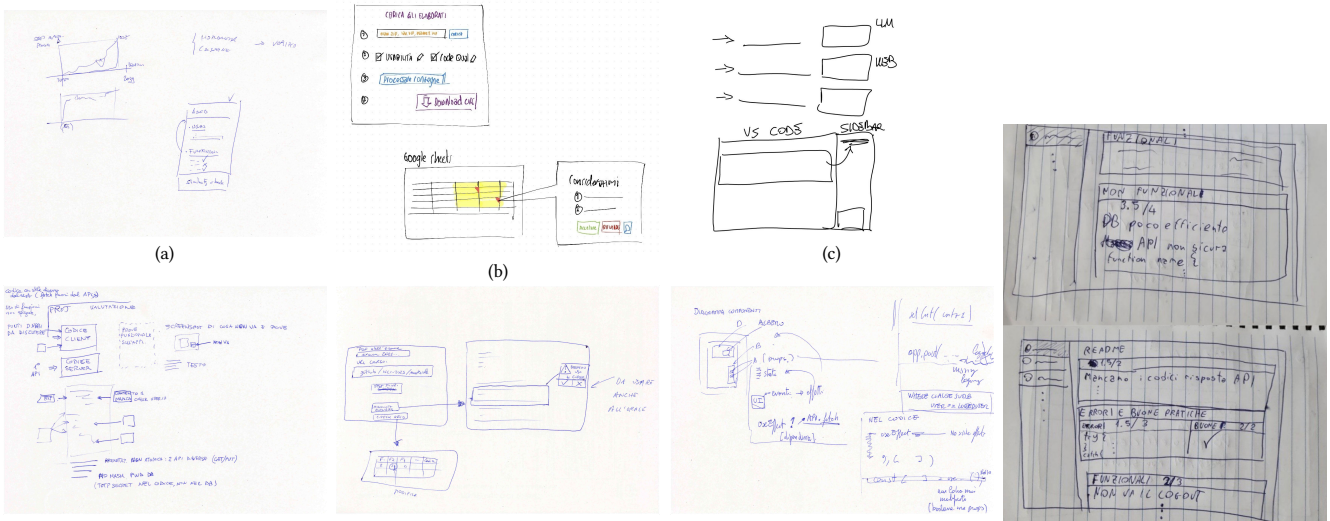


Figure 1: Instructors envisioned several types of assessment-support systems. P3 (c) proposed a tool directly integrated into the IDE they already use. P1 and P6 (a, g) sketched a requirements checker that flags unmet requirements with brief annotations. P4 and P7 (d, f) imagined inline support that highlights issues in the code and explains them. P2 and P5 (b, e) proposed a system that automatically populates an editable rubric with links to the relevant errors/evidence.

AI-based automation tools, underlying that it is welcome when it **supports** assessment work but without **replacing** instructors' judgment.

Instructors identify **pre-checks and completeness** as suitable for automation. P3 imagined rapid screening of required artifacts: *"it reads the README and tells me right away whether it's okay or not"*. These preferences were echoed in the sketching exercise: both P1 and P6 structured evaluation into explicit sections (e.g., README, functional/non-functional, errors and good practices) to make coverage visible (Figures 1a and 1d).

Similarly, instructors found that a possible automation is to extract **evidence** to drive oral probing. P4 paired screenshots of issues with textual explanations (Figure 1e). Along the same line, P7 provided an example in which the tool highlights an error in the code and explains why it is considered an error (Figure 1g). P2 and P5 took this further (Figure 1b and Figure 1f), proposing pre-compiled grids with tentative scores that instructors can accept or revise. P2 sketched an explicit approval workflow where the system proposes items, but the instructor must *"accept it, reject it or add some further clarification"* before suggestions become part of the assessment record (Figure 1b), underlining that grading is an instructor responsibility: *"[the instructor] gives the final grade. So it depends entirely on me"*. A similar pattern-starting from an initial score and then modifying it-also appears in P5's sketch 1f. P6 likewise emphasized the authority of decision: *"in the end, the last word is mine but it would be very convenient for speeding up the time"* further arguing that a tool cannot realistically be *"100% faultless"* (P6). P3 further reinforced this orientation by locating assistance inside the instructor's existing workflow (i.e., an IDE/VS Code sidebar), positioning the tool as an in-context companion rather than an autonomous grader (Figure 1c), underling that *"for now, the human is still necessary"*, particularly when grading requires interpreting intent, design rationale, or course-specific expectations.

3.3 Discussion on implications

Building on the challenges and needs identified in our themes, we translate instructors' perspectives into concrete implications for assessment-support tools. Tools should support artifact-consistency checks that cross-verify documented features, endpoints, and points mismatches as review items (Section 3.2.1). As reported in Section 3.2.2, because courses differ in stacks, requirements, and what instructors consider important, tools should be configurable at the level of checks. They should offer a check builder that lets instructors define and maintain course-specific validations (e.g., required artifacts, rubric-linked quality heuristics) and then share these checks across the grading team. In addition, tools should provide mechanisms to align instructors on what to consider important and need careful evaluation, by indicating when a particular issue has already been evaluated elsewhere (e.g., in another submission), enabling more informed and comparable decisions. As highlighted in Section 3.2.3 and Section 3.2.4, oral examinations are often driven by evidence found in the submitted project. This suggests that tools should summarize potential issues and pair each one with the evidence needed for rapid inspection, clearly stating what was detected, where it appears, and how it maps to the rubric criteria. Additionally, from Section 3.2.4 we found out that tools should support pre-filled rubric entries and possibility to accept/modify/reject at a fine granularity. This approach keeps instructors accountable for final judgments while still benefiting from time-saving assistance.

4 Conclusions and Future Work

In this paper, we presented a study with seven instructors who assess web applications developed as take-home project followed by an oral discussion. Through semi-structured interviews and a sketching exercise, we characterized the assessment workflow and surfaced recurring challenges around (i) using the README as the "entry point" while still needing to verify consistency between documentation and implementation, (ii) maintaining consistency under fatigue and across multiple graders, (iii) handling increased opacity when students rely on GenAI without critical understanding. Building on these insights, we explored with instructors how they envision tools that could support their current practice.

In future work, we plan to develop a working prototype and evaluate it with instructors in realistic assessment scenarios, using iterative testing to refine both the design implications and the tool itself.

Acknowledgments

This work was supported by the Cineca consortium.

References

[1] Kirsti M Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15, 2 (2005), 83–102. arXiv:https://doi.org/10.1080/08993400500150747 doi:10.1080/08993400500150747

[2] Paramasiven Appavoo and Anuja Meetoo-Appavoo. 2022. eExam Framework for Programming Classes. In *2022 3rd International Conference on Next Generation Computing Applications (NextComp)*. 1–6. doi:10.1109/NextComp55567.2022.9932218

[3] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 500–506. doi:10.1145/3545945.3569759

[4] John Biggs. 2001. The reflective institution: Assuring and enhancing the quality of teaching and learning. *Higher Education* 41 (2001), 221–238.

[5] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. doi:10.1191/1478088706qp063oa

[6] Zixin Chen, Jiachen Wang, Yumeng Li, Haobo Li, Chuhan Shi, Rong Zhang, and Huamin Qu. 2025. CoGrader: Transforming Instructors' Assessment of Project Reports through Collaborative LLM Integration. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25)*. Association for Computing Machinery, New York, NY, USA, Article 129, 18 pages. doi:10.1145/3746059.3747670

[7] Edoardo Cipriano, Alessio Ferrato, Carla Limongelli, Daniele Schicchi, and Davide Taibi. 2025. Leveraging Large Language Models to Assist Teachers in Code Grading. In *Artificial Intelligence in Education - 26th International Conference, AIED 2025, Palermo, Italy, July 22-26, 2025, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 15880)*. Springer, 204–217. doi:10.1007/978-3-031-98459-4_15

[8] Marcelle Droulers, Amita Krautloher, and Saeed Shaeri. 2026. Interactive oral assessment: co-existence of formative and summative purposes. *Teaching in Higher Education* 31, 1 (2026), 67–85. doi:10.1080/13562517.2025.2549945

[9] Simpson C. Gibbs, G. 2005. Conditions Under Which Assessment Supports Student's Learning. *Learning and Teaching in Higher Education* 1 (2005), 3–31.

[10] Suhas Kannam, Yuri Yang, Aarya Dharm, and Kevin Lin. 2025. Code Interviews: Design and Evaluation of a More Authentic Assessment for Introductory Programming Assignments. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS '25)*. Association for Computing Machinery, Pittsburgh, PA, USA, 554–560. doi:10.1145/3641554.3701806

[11] Sam Lau and Philip Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 106–121. doi:10.1145/3568813.3600138

[12] David López, Josep-Llorenç Cruz, Fermín Sánchez, and Agustín Fernández. 2011. A take-home exam to assess professional skills. In *2011 Frontiers in Education Conference (FIE)*. F1C–1–F1C–6. doi:10.1109/FIE.2011.6142797

[13] Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaojing Shi. 2024. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Trans. Comput. Educ.* 24, 1, Article 10 (Feb. 2024), 43 pages. doi:10.1145/3636515

[14] Ed Novak and Peter Ohmann. 2023. Oral Exams in CS-Education: Pros and Cons in the Age of AI Assisted Programming. *J. Comput. Sci. Coll.* 39, 3 (Oct. 2023), 32–33.

[15] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. Comput. Educ.* 22, 3, Article 34 (June 2022), 40 pages. doi:10.1145/3513140

[16] Matthew Peveler, Evan Maicus, and Barbara Cutler. 2020. Automated and Manual Grading of Web-Based Assignments. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 1373. doi:10.1145/3328778.3372682

[17] Antonio Carvalho Siochi and William Randall Hardy. 2015. WebWolf: Towards a Simple Framework for Automated Assessment of Webpage Assignments in an Introductory Web Programming Class. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 84–89. doi:10.1145/2676723.2677217

[18] Michael Striewe and Michael Goedicke. 2014. A Review of Static Analysis Approaches for Programming Exercises. In *Computer Assisted Assessment. Research into E-Assessment*, Marco Kalz and Eric Ras (Eds.). Springer International Publishing, Cham, 100–113.

[19] Chenyan Zhao, Mariana Silva, and Seth Poulsen. 2025. Language Models are Few-Shot Graders. In *Artificial Intelligence in Education - 26th International Conference, AIED 2025, Palermo, Italy, July 22-26, 2025, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 15880)*. Springer, 3–16. doi:10.1007/978-3-031-98459-4_1

A Interview set of questions

Below we report the initial set of questions.

(1) What course are these projects for?
(2) What are the main topics covered in the course?
(3) What is your role in grading these projects?
(4) How many years of experience do you have in grading these projects?
(5) How many students do you grade each session?
(6) How many people grade (you, TA, …)?
(7) Can I see a project?
(8) Can you show me the workflow when assessing the exam starting from the delivery up to giving the mark to the student?
   (a) How is composed the grade?
   (b) Is there a rubric to follow?
   (c) When designing the rubric, how do you decide the maximum point value for each criterion?
   (d) During the assessment, how do you decide how many points to give for each criterion (e.g., give 0.5 out of 1)?
(9) What are the top 3 challenges when assessing the exams? What are the three easy things?
(10) When grading projects for this or similar exams, do you use any automation tools? Why or why not?
(11) How could you be supported in improving the phase of grading projects? You are allowed to come out with whatever solution you could think of, even if not feasible for you?
(12) Can you translate your solution into a sketch on a sheet of paper?