

How Computer Science Teachers Assess Open-ended Programming Assignments

Arthur Rump
University of Twente
Enschede, The Netherlands
arthur.rump@utwente.nl

Angelika Mader
University of Twente
Enschede, The Netherlands
a.h.mader@utwente.nl

Vadim Zaytsev
University of Twente
Enschede, The Netherlands
vadim@grammarware.net

Abstract

We studied how instructors assess open-ended programming assignments in university-level computer science courses in the Netherlands. Evaluating such assignments is labour-intensive, so our long-term goal is to develop tool support for this assessment. Effective tool support, however, requires understanding the existing practice. Through interviews with teachers and teaching assistants and analysis of their rubrics and grading guidelines, we characterise the topics and skills instructors assess, how they structure and use grading criteria, and how they experience the assessment process. Our findings provide an empirical basis for designing tool support for evaluating open-ended assignments.

CCS Concepts

• **Social and professional topics** → **Student assessment**; *Computer science education*; *Computer engineering education*.

Keywords

programming education, open-ended assignments, assessment, rubrics

ACM Reference Format:

Arthur Rump, Angelika Mader, and Vadim Zaytsev. 2026. How Computer Science Teachers Assess Open-ended Programming Assignments. In *Proceedings of 26th Koli Calling International Conference on Computing Education Research (Koli Calling '26)*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/XXXXXXX.XXXXXXX>

1 Introduction

In open-ended programming assignments we ask students to work in a problem space where multiple solutions are defensible. Rather than just showing that they can produce a prescribed output, students may need to make, justify and document design decisions, integrate programming and software engineering practices, and develop a substantial artefact. Such assignments can therefore provide evidence of learning that is difficult to obtain through short, closed exercises or examinations. Larger programming tasks are, for example, better suited to assessing structural design choices [13].

The same openness that makes these assignments educationally attractive also makes them challenging when we come to assessment. Solutions *will* differ in functionality, architecture, and process,

and assessors cannot simply compare them to a single model solution. They must determine which aspects of the work constitute relevant evidence, articulate standards that apply to all solutions, and exercise judgement when predefined criteria do not fit. In larger courses, these judgements may be distributed across multiple teachers and teaching assistants, leading to additional concerns about workload, consistency, transparency, and the defensibility of grades. Previous work has shown both that assessing substantial computing projects requires considerable staff effort [12, 24] and that human graders may apply programming-assessment criteria inconsistently even when using a rubric [20].

We investigate why instructors use open-ended assignments (RQ1); what they assess and what evidence they use (RQ2); how assessment guidelines are structured (RQ3) and used (RQ4); and how instructors experience the assessment process (RQ5). To answer these questions, we interviewed teachers and head teaching assistants responsible for 12 programming and software engineering courses at seven Dutch research universities and analysed the rubrics and assessment guidelines used in those courses. The courses cover different stages of the computer science curriculum and different forms of open-endedness, ranging from assignments with prescribed functionality but freedom in implementation to projects in which students define their own product or work for an external client.

This paper makes three contributions. First, it provides an empirical characterisation of what is assessed in open-ended programming assignments. Second, it characterises the different forms that assessment guidelines take and how they support grading, feedback, assessor coordination, and communication with students. Third, it identifies recurring tensions in these practices, such as consistency versus professional judgement, transparency versus checklist behaviour, and comprehensive criteria versus manageable assessment, that provide requirements for future assessment research and tool development.

The remainder of the paper reviews relevant work and defines the terminology used in this study (section 2), describes the study methodology (section 3), presents our findings (sections 4–5), and discusses their implications, limitations and directions for future work (section 6).

2 Background

Assessment is a complex topic, with many different aspects to consider [15]. In this section, we give a short introduction on the topic, with a focus on assessment strategies, grading guidelines and rubrics, and open-ended assignments, particularly as applied in programming education.

2.1 Assessment

In general, we can distinguish two types of assessment: formative and summative. *Formative assessment*, also called *assessment for learning* [33], is about getting feedback that supports learning [29]. *Summative assessment*, on the other hand, happens after learning, is used to make decisions (for example whether a student has passed a course) and typically involves giving grades [3]. In this study we mainly considered assignments that have a summative purpose, though many include formative aspects as well.

For summative assessment, grades can be determined in two ways: norm-referenced, where students are ranked and receive a grade based on their position in the ranking, and criterion-referenced, where students are assessed against the intended learning outcome or criteria derived from those. In order to maintain constructive alignment between intended learning outcomes, teaching and learning activities, and assessment tasks, the latter model is recommended [3]. In the Netherlands, this is the strategy that is typically used for summative assessments.

In a criterion-referenced system, there are two main strategies for setting criteria and determining grades. In a *holistic assessment*, criteria describe the quality of a work as a whole, and an instructor determines a single grade based on those descriptions. In an *analytic assessment*, a work is assessed on different aspects, with criteria given for each, and a grade is typically calculated as a weighted average of subgrades given for each aspect [3]. In both cases, it is helpful to clearly describe the standards students should aim for. The final grade can also be determined holistically even with an analytic rubric [32].

2.2 Rubrics

The term *rubric* is widely used in practice, but it does not seem to have a clear meaning in that use, often referring to any type of description or guideline used for grading or assessment [7]. In a stricter definition, a rubric needs to include multiple criteria and gradations of quality for each criterion, describing what is required to fulfil that criterion at each level [11]. This is typically formatted as a table with criteria in the first column and additional columns for each level. Another popular definition additionally requires a scoring strategy, which may be holistic or analytic [22]. In this paper we will refer to a rubric according to the first definition, and use assessment guidelines or grading guidelines as the more general term.

Rubrics, in this stricter sense, are often recommended as a tool for assessing complex assessment tasks [3], with more standardised, objective and transparent assessments as the main benefits [6, 14]. Designing a good rubric can be challenging, though, with too vague descriptions as a common fault [6]. Besides a grading tool, rubrics are also a teaching tool, for example to facilitate feedback, whether by an instructor, a peer, or the student themselves, at least if students learn how to use the rubric [2, 23]. Although evidence that rubrics improve self-regulation is mixed, clear learning goals, criteria and level descriptions may support students' learning, and rubrics are a common way to achieve this [4].

2.3 Open-ended assignments

In programming education, both smaller exam-type questions and larger programming tasks are commonly used for assessment and can cover the typical intended learning outcomes of an introductory programming course. However, larger programming tasks are known to be more suitable to assess structural design choices [13]. There are suggestions that open-ended assignments might improve students' self-efficacy in programming [30] and that an approach where students are encouraged to tinker has positive effects [19].

With open-ended we mean that there is no definite endpoint for an assignment, no single model solution that is the correct solution. This is also one of the defining features of an ill-defined problem or domain [18]. A programming assignment can be ill-defined because it, for example, requires students to make design choices for a complex problem, resulting in many different good solutions, or because the assignment leaves it up to the students what they are going to make or which problem they will try to solve [27].

Open-endedness is a feature seen in different forms of learning based in the constructivist tradition, like in project-based learning, where the open-ended problem is its driving question [17].

2.4 Assessment in programming education

When it comes to assessment in programming education, there is a large body of work on automated assessment [16], but assessment practices in general get less attention in the literature. Messer et al. [20] studied the consistency of graders in grading an open-ended programming assignment, among other reasons also aiming to find a stable base for the evaluation of automated assessment tools. They had graders follow a rubric covering correctness, code elegance, readability and documentation, but found that consistency of the assessments was very poor. Brown et al. [5] investigated the use of comparative grading as an alternative to the traditional rubric-based approach on the readability aspect and found no strong correlation between the results from that approach and the traditional grading, concluding that grading readability is very difficult.

Fitzgerald et al. [10] let four instructors grade 20 student solutions to a programming problem, without prior discussion of grading criteria or rubrics, to examine how instructors approach this grading. They found that two of the instructors created a set of criteria and used an analytic approach to come to a grade, while the other two primarily aimed to assess whether the student was competent enough at programming to move to the next class, performing a more holistic assessment towards that question. Statistically the four graders were consistent, though in some cases the different approaches led to very different outcomes for a single program.

Riese and Bälter [24] interviewed instructors of non-CS CS1 courses about their perspectives on assessment. All courses had a similar assessment setup, with lab assignments, a mid-term exam (in some cases open-book) and an individual final project, where students can choose from a pool of project descriptions. The instructors considered the project to have both formative and summative aspects, with teaching assistants having a big responsibility in giving feedback and guidance for the formative aspects and doing the grading for the summative.

Stegeman et al. [31] designed and evaluated a rubric on code quality in programming courses, covering topics like names, comments, formatting and indentation, and decomposition. In a rubric for automatically assessing App Inventor projects, Alves et al. [1] tend to focus more on the use of various constructs, like operators, variables, lists, loops and conditionals. The lowest performance level indicates that a construct was not used, while higher levels describe more advanced uses of each construct, like a single if-statement, an if-else-statement and multiple if-else statements. Neutens et al. [21] interviewed teachers in secondary education about the criteria they use for assessing code, categorising them as function (including correctness and usability), readability, use of constructs, algorithmic thinking, testing and debugging, mathematical concepts (boolean logic and arithmetic), creativity and personal aspects (like behaviour in class). Functional correctness was considered one of the most important aspects by most teachers when asked to rank the various criteria. For some teachers the primary focus was on the correct use of constructs, assessing if the student understood how the constructs required for the exercise work, rather than the correctness of the complete program.

Computer science capstone courses often involve programming or software engineering practice, and thus show many similarities with open-ended programming assignments. In these courses it is common to also assess skills like teamwork and communication, sometimes looking into version control history or using mentor observations to inform the assessment [12]. However, capstone projects exhibit enough traits exclusive to them to warrant a separate investigation. While we acknowledge the similarities, the rest of the paper deliberately covers the broader category of open-ended assignments, which includes, but is not limited to, capstone projects.

3 Methodology

We contacted course coordinators listed in public repositories of courses in computer science and closely related programmes for all public research universities in the Netherlands. We selected courses based on their topical relevance (programming or software engineering) and read the course descriptions to understand the assessment practices. We selected courses where projects or assignments are a major part of the summative assessment practice. In total, we contacted a teacher or coordinator for 31 courses, resulting in 11 interviews with a teacher or head teaching assistant for the course.¹

In the interviews we covered 12 courses (in one interview two courses were discussed) from 7 research universities in the Netherlands. All courses are primarily part of a CS programme, 11 at the Bachelor level and 1 at the Master level. From the Bachelor-level courses, we covered 6 first-year courses and 5 advanced courses. One of the first-year courses is the first introduction to programming in its programme; for the others some prior programming experience is expected. Four courses have fewer than 50 students, one course between 50 and 100, five courses between 100 and 200, and two courses have more than 200 students.

The courses teach a variety of programming languages: in four courses students are free to choose the language they will use, but where a language is prescribed, C#, Haskell, Java, JavaScript and Python are each used in two courses, followed by Agda, C++, Erlang and Kotlin each used in one. Note that some courses cover more than one language, so the total count adds up to more than 12. Looking at the course goals, seven courses aim to give students *broadened programming skills* and five courses teach *software project engineering*. One course covers *programming paradigms*, particularly *object-oriented programming*, *functional programming* and *distributed programming*; among which *object-oriented programming* and *functional programming* are also the specific topic for one other course each. The topics *software design*, *web* and *databases* are each covered in three courses and *games* and *formal methods* are each covered in one course.

All courses we covered are open-ended, but in different ways: some have strictly specified functional requirements, but give students freedom to choose their implementation approach; others let students work on their own ideas or on a project for an external client. Even when requirements are specified, many courses offer bonus points for extra functionality, either from a given set of options or fully up to the students. Some courses apply restrictions other than functional requirements, such as certain technical requirements, letting the students start with a given template or even guiding them through incremental steps. We see this diversity as a good coverage of the spectrum of open-ended assignments and different types of courses.

For each course, we asked the teacher to share their rubric or assessment guideline and assignment descriptions with us. We used these in our qualitative analysis, outlined below, and also to get an overview of the course, the assignment and the assessment in preparation for the interview. Each interview was conducted by the first author, who is a PhD student at a Dutch university with a background in both university level and high school level computer science education. All interviews took place between October 2025 and February 2026, online through Microsoft Teams, lasting around 45 minutes each. The interviews were recorded, which we used to manually correct the automatic transcript created by Microsoft Teams. After this step, the recordings were deleted and personally identifiable information removed from the transcripts.

The interviews were conducted in English or in Dutch, depending on the preference of the interviewee. For the analysis, we kept transcripts and provided documents in the original language, as all participating researchers are proficient in both languages. All quotes in this paper have been translated to English by the first author, occasionally using the DeepL translation tool to find the best English equivalent to expressions.

For analysis of the transcripts and assessment guidelines, we used a two-step flexible coding process [9], which is particularly suited for interviews in which different topics or aspects of a topic are discussed. The interviewer created an index-coding in a coding tool [25], marking the topic of each section of the interview transcripts, closely related to the interview questions. For each topic, he then went through the transcripts, filtered to only the sections related to that topic, and analytically coded in two passes: in the first pass creating new codes as aspects were identified in the data and in the second pass combining codes that relate to general

¹This study was approved by the Ethics Committee Computer and Information Science of the University of Twente, application nr. 251783.

concepts and generally refining the coding. For the topics of assessment guideline structure and content, we also coded the assessment guidelines themselves, alongside the interview transcripts.

The codebooks were then shared with the other authors, who both went through disjoint subsets of the transcripts, for 3 and 4 cases respectively, to apply these codes to the relevant sections, also working topic-by-topic. We compared the resulting codings and discussed all conflicting selections, which had different codes applied by different coders, and lonely selections, which had a code applied by only one of the coders. Based on this discussion, we updated the description of a few codes and the main coder went through the other transcripts once more to check that those codes were applied correctly. In many cases, a lonely code was also applied to a different selection in the same transcript by the other coder, so this did not meaningfully change our results. The final codebooks, along with the general interview guideline, are available in the supplementary material [26].

In the following sections we will discuss the results of this analysis, describing the different **themes** we found in both the transcripts and in the assessment guidelines, with supporting “quotes” from an interview or assessment guideline, always attributed to its case number. A full overview of which themes apply to which cases, including all quotes referenced in this paper, is available in the supplementary material [26]. We first discuss the question of using open-ended assignments in [section 4](#), then describe the assessment guidelines and their use in [section 5](#).

4 Why use open-ended assignments? (RQ1)

All courses in scope have open-ended assignments, albeit for a different definition of open-endedness, as covered in the description of the courses in [section 3](#). While we have given some general reasons why instructors might choose open-ended assignments in [subsection 2.3](#), in this section we further argue their relevance through the reasons our respondents have given for choosing an open-ended assignment in their course. One teacher answered they do so “because I want to make my life difficult, sometimes, I feel” (11), though there are several factors that might make it worth the effort. Figure 1 gives an overview of the responses and how many participants mentioned each aspect.

Seven teachers mentioned that they want to give their students a **realistic**, practical experience: “the goal is [to] prepare them for reality” (06) and “I think it also helps to keep it fresh, so to keep including new technologies in [the course]” (04). This is related to the idea that the assignment is not just for summative assessment, but it should also be a **learning experience**: “I think it’s also very good as a sort of learning experience, especially if you’re not yet working in IT, to see that what you planned to get done, that turns out to be a bit of a let-down in the end.” (04) One respondent described the importance of letting students experience the struggles of project management: “We give them two lectures on Scrum with the same content. The first Scrum-lecture we tell them how it all works. The second Scrum-lecture happens after the first sprint. And we give them the exact same information again, just expressed differently. And I often got the comment: ‘Why didn’t you tell us earlier?’ I did, but now you know. So show, don’t tell. That is everything here.” (06)

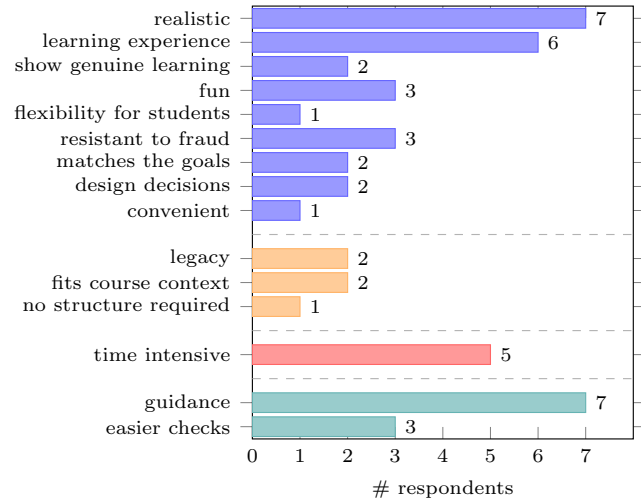


Figure 1: The number of respondents (out of 11) that mentioned each aspect of using open-ended assignments. The four sections show perceived benefits of open-ended assignments, other considerations, perceived downsides of open-endedness and perceived benefits of structure, respectively.

Besides learning while working on the assignment, two respondents mentioned that they believe an open-ended assignment can **show more genuine learning**: “the more you make it critical and open, the more you learn about genuinely [...] what the student learned.” (02) Respondent 07 argued that this is hard to achieve with an exam: “on an exam of at most 3 hours [...] you don’t have the space to give them a larger assignment in which you can really sense what they are capable of. [...] I can test that they get the hang of the technique, but not that they can apply it in a larger project, so that’s why I think the project is very important.” (07)

Others mention that open-ended assignments are more **fun**: “Students often look forward to it very much, when it’s an open-ended assignment. It provides extra motivation.” (03) It also gives some **flexibility to students**, for example to dive deeper into the parts of the subject that interest them most: “[...] to allow the students to grow in different directions. So it gives you this customizability, right? Like a student maybe wants to learn more about, I don’t know, game physics. [...] Then you allow them to, with the bonuses, you allow them to do a new concept, create it from scratch. And then you have another student that just wants to do the typical power up, they don’t care so much about this aspect. Then again they have the freedom.” (02)

Respondents also mentioned that open-ended assignments can be more **resistant to fraud**: “I could make a closed assignment, but then a sort of standard will emerge. Or I need to revise the assignment every year. Or a sort of best solution emerges. That seemed like a dangerous situation to me. [...] Then that solution is bound to get passed around, of course, so that doesn’t seem ideal to me.” (04) With the rise of generative AI, that has also become a concern: “What I’m really afraid of with giving too much structure and templates, especially nowadays, is that you just put it into ChatGPT and then you say ‘do it’ and then the more you give the structure, the more it’s

like accurate.” (02) Respondent 10 also mentioned seeing a shift in recent years from closed homework assignments to a mix of exams and more open-ended assignments.

In some cases an open-ended assignment **matches the goals** of a course particularly well: *“The openness of the assignment matches the openness of the outcomes.”* (11) In their case, one of the course goals is that students learn to make **design decisions**: *“They need to make a decision. They need to own it, right? Like, I selected the stack. You didn’t tell me to use the stack. I selected the stack. And therefore, it’s also my fault. Also my triumph, right, if it works great. If it doesn’t work, you know, I need to take into account what I could do to improve the whole thing.”* (11) For respondent 05 this is part of giving a realistic assignment: *“To show that there are different design choices, there are alternatives. Then it’s a matter of weighting and looking at the pros and cons of each design choice and then to report it.”* (05)

For one course, the instructor simply found it more **convenient** to use an open-ended assignment to achieve the goals of the course: *“Of course you can also do that with something that’s not your own idea, but then you often need a stakeholder to see what they think of it. And now it’s more: ‘What do I think we need for this?’”* (03) Letting students take ownership of these decisions made the course easier to organise.

Finally, teachers also have practical reasons for working with open-ended assignments. For some, it is part of the **legacy** of a course they inherit: *“this was the concept of the course from the beginning,”* (06) and *“this was already the case when I followed the course, some eight years ago.”* (03) It can also be the case that an open-ended assignment **fits the course context**, for example to prepare students for a larger *“software project where they work with a group of 10 for a client. This way we have a line in the bachelor where they first do our [smaller] project and then have a big project at the end.”* (03) One teacher also noted that there is **no structure required** for their course: restricting students is not necessary to assess their achieved learning outcomes.

4.1 Caveats and considerations

While there are many reasons an open-ended assignment could be a good match for a course, that might not always be the case. The one downside mentioned by our respondents is the **time-intensive** nature of open-ended assignments, both for students and teachers: *“setting up the course doesn’t take much effort, but the supervision [of students] simply takes a lot of time.”* (04) Respondent 08 mentions their assignments are a bit more structured, because *“these are assignments for which they don’t have a lot of time. In general these are assignments they work on for a maximum of two weeks.”* (08)

In some cases, teachers choose to provide structure for the students, to some degree limiting the openness of the assignment. They do this, for example, because students need, or at least benefit from, receiving **guidance**. This is especially the case for junior students or in cases where they have not yet acquired certain skills: *“I think one factor is that these are first-year students, and that’s why a slightly more hand-holding approach was chosen,”* (08) and *“I feel like it’s a good strategy for a first year course, first study period, because imagine now if I did completely open, students would be completely lost, right?”* (02) A few teachers also mention that structure gives

them something to hook into for assessment, to have **easier checks**: *“That’s where it [openness] collides with having structure to check things, right? To do unit testing or ...”* (02)

This shows that there is a trade-off between structure, which may guide students to a more specific experience, and openness, which allows for a richer and more realistic learning experience. In the courses we cover we see teachers tending towards the open-ended side of this balance, despite the major downside of the extra time it may take. As much of this time is spent on assessment, both formative and summative, this confirms our motivation to support these processes with tooling, making them more practically viable. In the next section, we will discuss the assessment practice of these courses, to better understand the context in which such a tool should work.

5 An overview of assessment guidelines

To get an overview of how instructors assess open-ended programming assignments, we asked our respondents to share the assessment guidelines for the courses they teach and in an interview we asked them how their course is set up, how they use the assessment guidelines and their experience with the assessment process. In the following sections, we discuss the topics that are assessed ([subsection 5.1](#)), the structure of the assessment guidelines ([subsection 5.2](#)), how the assessment guidelines are used ([subsection 5.3](#)) and the teacher’s experience with that assessment ([subsection 5.4](#)). In these sections, we will describe the results of our analysis with themes and supporting quotes. In [section 6](#) we will then draw further connections among these aspects and compare our results with the existing literature.

5.1 What is assessed? (RQ2)

Of the courses we looked at, four include one or more programming assignments, which are intended purely to test programming skills. Three of these combine this with an additional written exam. The other eight courses use a single integrated project, without clearly defined subtasks and which usually assesses skills beyond just programming. Different assessment methods are applied for these, including assessing the final product, presentations, meetings with students and teacher-observation of their process, written reflections, a peer review, a report and evidence of individual contributions.

Even though not all projects specify the functionality students should implement, **functionality** is still assessed in one way or another in all assignments. For some these are detailed requirements (05) or input/output relations specified by tests (07, 08). For others, it is described in more general terms like *“app does not function as described in the specification”* as a descriptor for insufficient to *“app functions more or less without errors”* for sufficient (04), where the specification is given by the students themselves. Two courses include the option of an **open bonus** in their assessment guideline, like *“You’ve added fun extras that required challenging programming.”* (02)

Most projects, with three exceptions, also assess **code quality**. For some, these are formulated as goal-oriented criteria, like *“The software is reliable and maintainability-aspects are clearly applied.”* (01) and *“The source code is easily readable.”* (01) Others use more

specific requirements around **code style**, like “Comments are missing in places where they are needed.” (06), “The variable names are very cryptic. One-letter variable names should be avoided, except for counters in for-loops.” (02), or “Clear naming, structure and error handling.” (04) A few assignments also assess the use of **design patterns**, either a specific pattern students are supposed to use (“not adhering to MVC (by having an association from model to gui)” (05)) or in more general terms (“The implementation selects the appropriate design patterns and rationalizes their adoption” (11)).

Design is assessed in 8 out of 12 assignments, either as architecture descriptions or as it is implemented in the code: “You make logical use of member variables, method parameters, and/or properties to give objects/classes access to each other.” (02) as a positive descriptor, “Monolithic code with no packages, big classes poorly related to concepts and no documentation.” (09) as an insufficient descriptor. Respondent 05 said they “try to see whether students have good modelling skills,” (05) which they assess mainly based on the diagrams students create of their design. For use case diagrams “they are supposed to not use technical terms, it should be kept kind of abstract, language independent,” (05) while in communication and class diagrams “we expect more technicality, association, division of responsibility.” (05) This should translate to the code students write, where they look at “for example, having small, well-focused classes.” (05)

For a few assignments, the teachers also assess if students are able to use the appropriate **code constructions** in the language, ranging from if-statements or loops to inheritance or constructions for concurrency: “we’re trying to see if they understood the main concepts, like for-loops, recursion, inheritance, [...] up to interfaces and abstract classes” (02), or “whether they use prepared statements or not, whether they handle concurrency appropriately.” (09) This can also be more open: respondent 04 wants their students to generally learn new techniques and assesses that “correct programming techniques are used with the required mastery in an efficient manner.” (04)

In about half of the assignments, **testing** is assessed. This may include different types of testing: the rubrics of case 09, for example, rate “Applied a testing strategy with Unit tests, Integration tests and User tests.” as above average, and applying two out of those three as sufficient. Others also describe quality attributes of the tests, like “The testing methods cover all code and are automated. The approach is exceptionally well-reasoned and structured. The tests comply with all points set out in the unit testing guidelines.” (01), or the testing process, like “They started testing early (sprint 1 or 2).” (06)

Besides the product and the code, the **process** is also an often-assessed aspect, including aspects of collaboration, but also the use of tools and infrastructure, like version control and CI/CD pipelines, to enable collaboration. The assessment guidelines of case 04 cover all in a single criterion “CI/CD, version control and collaboration,” (04) for which the highest level (good) reads “Every component its own issue, on its own branch, with explanations and comments on issues and code. Clear commit messages. Have mastered deployment.” (04) Other cases include more thorough descriptions of their expectations for teamwork, for example including criteria on “Reliable presence in the group common room”, “Attendance of group meetings”, “Participation in group meetings”, “Attitude in group meetings”, “Attitude in the development”, “Working as a team”, “Fair division of labour”, “Pair programming” and “Helping others” (06).

In four of the assignments, students are asked to create a **product vision**, with requirements, stakeholders, user stories and/or scenarios for their product. This is typically assessed on relevance and completeness: “The product vision is appealing and answers the what-, who- and why-questions. Personas are clearly described and match the target audience. Features are independent, coherent, relevant.” (01) “Description and use cases: complete, detailed, aligned with the rest; Features, MoSCoW and components: matches description, appropriate and specific” (03)

In three assignments students are asked to **reflect** on their work, sharing what they learned and what they would change in a next iteration, regarding the process, collaboration and/or technical decisions. Case 04 covers all of these: “Reflection shows insight and gives a clear overview of important design and implementation decisions, with possible alternatives, whereby disadvantages and advantages of the choices made are discussed. There is also a reflection on the collaboration process.” (04)

While written reports or presentations are used to gather information to assess any of the above topics, in some cases the **reporting** itself is also assessed: “General polish, language use, and images: clarity, readability, visual appeal” (03) Respondent 09 mentions that “the final presentation is mostly related to the actual presenting skills rather than the actual functionality of the project.” (09)

5.2 How are assessment guidelines structured? (RQ3)

Figure 2 shows an overview of the different structures used in the assessment guidelines in this study, along with the number of courses that use such a structure. We describe two axes in the figure: the format of the assessment guidelines as either a **table** or a **list** and how the guidelines describe different levels of achievement, using **level descriptors** or **marks**, or if they only give the **plain criteria** without describing levels of achievement.

Four assessment guidelines make use of a traditional **rubric**: a table with levels of achievement on the columns, criteria on the rows and level descriptors in each cell. In this form, the **same levels** are typically applied to each criterion. Another three cases also use level descriptors, but write them as a list, rather than a table. In all those cases, **different levels** are used for each criterion.

An alternative to level descriptors is the use of **marks**. This represents a more analytic approach to assessment, where positive and negative aspects are counted and summed together. This is often used for exam questions, where points are awarded for parts of a correct answer or steps in a calculation, but it can also be applied to larger projects to determine a level of achievement.

Finally, three courses in our study do not specify levels of achievement and only give the **plain criteria**, at least for some of them. In one case, this takes the form of an **empty rubric**, where a table is used to generally describe different levels and the criteria, but the cells do not contain specific descriptions for each criterion at each level. Two other courses simply list the criteria with a maximum number of points, leaving it up to the assessor how many of those points to award in each situation.

The total number of cases in Figure 2 exceeds the total of 12 courses, because some combine multiple approaches for different aspects of the assessment, like including both level descriptors and

	Level descriptors	Marks	Plain criteria																																																		
Table	<table><tr><th></th><th>Insufficient</th><th>Mediocre</th><th>Sufficient</th><th>Good</th></tr><tr><td>App (15%)</td><td>App doesn't work, no description in the specification</td><td>App works mostly as described in the specification, but has minor errors, without full test coverage</td><td>App works more or less without errors, with good test coverage</td><td>App works more or less without errors, with good test coverage</td></tr><tr><td>Code style (20%)</td><td>Unclear naming, no clear function division, missing documentation</td><td>Unclear sections of code. For example unclear naming, no clear function division or documentation</td><td>Clear naming, structure and error handling with small hints in essential places</td><td>Clear naming, structure and error handling. Clear hints that clarify the structure of programs</td></tr><tr><td>CI/CD, version control and collaboration (15%)</td><td>No issues per component, no branch per issue, bad or missing commit messages, clear non-working code on master</td><td>Works on separate branches, but with bad or missing commit messages</td><td>Works on separate branches, with clear commit messages</td><td>Every component has its own issue, on its own branch, with partners and comments on figures and code. Clear commit messages</td></tr></table>		Insufficient	Mediocre	Sufficient	Good	App (15%)	App doesn't work, no description in the specification	App works mostly as described in the specification, but has minor errors, without full test coverage	App works more or less without errors, with good test coverage	App works more or less without errors, with good test coverage	Code style (20%)	Unclear naming, no clear function division, missing documentation	Unclear sections of code. For example unclear naming, no clear function division or documentation	Clear naming, structure and error handling with small hints in essential places	Clear naming, structure and error handling. Clear hints that clarify the structure of programs	CI/CD, version control and collaboration (15%)	No issues per component, no branch per issue, bad or missing commit messages, clear non-working code on master	Works on separate branches, but with bad or missing commit messages	Works on separate branches, with clear commit messages	Every component has its own issue, on its own branch, with partners and comments on figures and code. Clear commit messages	✕	<table><tr><th></th><th>Insufficient too much of the program is missing</th><th>Under par important aspects missing</th><th>Sufficient everything is there, but some issues</th><th>Good solid work, some minor issues</th><th>Excellent very high quality, several improvements</th></tr><tr><td>Can be executed, instructions</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Maximum explanations and descriptions in report</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Looks, feel and grade of the product</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Range and scope, quality, security</td><td></td><td></td><td></td><td></td><td></td></tr></table>		Insufficient too much of the program is missing	Under par important aspects missing	Sufficient everything is there, but some issues	Good solid work, some minor issues	Excellent very high quality, several improvements	Can be executed, instructions						Maximum explanations and descriptions in report						Looks, feel and grade of the product						Range and scope, quality, security					
		Insufficient	Mediocre	Sufficient	Good																																																
App (15%)	App doesn't work, no description in the specification	App works mostly as described in the specification, but has minor errors, without full test coverage	App works more or less without errors, with good test coverage	App works more or less without errors, with good test coverage																																																	
Code style (20%)	Unclear naming, no clear function division, missing documentation	Unclear sections of code. For example unclear naming, no clear function division or documentation	Clear naming, structure and error handling with small hints in essential places	Clear naming, structure and error handling. Clear hints that clarify the structure of programs																																																	
CI/CD, version control and collaboration (15%)	No issues per component, no branch per issue, bad or missing commit messages, clear non-working code on master	Works on separate branches, but with bad or missing commit messages	Works on separate branches, with clear commit messages	Every component has its own issue, on its own branch, with partners and comments on figures and code. Clear commit messages																																																	
	Insufficient too much of the program is missing	Under par important aspects missing	Sufficient everything is there, but some issues	Good solid work, some minor issues	Excellent very high quality, several improvements																																																
Can be executed, instructions																																																					
Maximum explanations and descriptions in report																																																					
Looks, feel and grade of the product																																																					
Range and scope, quality, security																																																					
	Rubric (4)		Empty rubric (1)																																																		
List	Communication between objects ✕ You have very few classes, so there is little communication ✕ Lots of communication through global or class-public variables ✓ Good use of method parameters for communication ✓ Uses parameter passing with minimal necessary information Comments ✕ Comments are missing completely ✕ There are comments, but not enough to make the code readable ✕ The code is overcommented, don't explain every detail! ✓ Generally good, but could be a bit more extensive ✓ Clear and informative comments, helps to understand structure	Code quality +1 Intuitive structure, well documented +1 Methods and functions are well-named +1 Good separation of concerns in classes -1 Multiple functions are too long -1 Comments are missing -2 Clearly copied code in the same file -1 Outcommented code is left behind	Code quality - Design: extensibility, documentation 3pt - Function: usage of methods, construction of methods, error handling 5pt - Project structure: project clarity, usage of variables, usage of comments 2pt - Quality assurance 1pt - Version control 1pt																																																		
	List of levels (3)	List of marks (4)	List of criteria (2)																																																		

Figure 2: The ways in which assessment guidelines are structured and how they describe levels of achievement. The number of assessment guidelines in this study that use this structure is indicated in parentheses. Some courses use more than one type of structure, so the total sums to more than 12. The example for *Rubric* is quoted from case 04 and those for *Empty rubric* and *List of criteria* from case 03. The examples for *List of levels* and *List of marks* are inspired by cases 02 and 06 respectively.

plain criteria in a list (08) or using marks for some criteria and level descriptors for others (06). Not included in the figure are **test cases**, which are also used for two assignments.

When it comes to grading, most assessment guidelines contain some evidence of **analytic grading** where a final grade is calculated based on scores assigned to each part. Indicators for this include numerical weights given to each criterion or a grade calculation formula in a spreadsheet. In one case the assessment guidelines explicitly call for a **holistic grading** approach: “For the grade you give a number based on your overall impression. You also assess individual parts, but you don’t give partial grades that somehow add up to a final grade.” (02)

5.3 How are assessment guidelines used? (RQ4)

The primary use case for the assessment guidelines seems to be summative **grading**: “we basically use the rubric [...] as a way of guiding the grading of the assignment.” (11) For the cases where test cases are part of the assessment guidelines, this is partially **automated**, but in most cases it is a manual effort.

Two respondents describe using a **holistic grading** approach: “I guess the point there is that you don’t just add up all these small pieces and then you’re just reaching a passing grade. Because a student could do tiny things that are not really showing that they learned about [t]he objectives of the assignment and then they can get a passing grade. [...] I think the phrase basically tries to avoid that.” (02)

Others typically do **analytic grading**, where, for example, “for each one of the criteria [...] you’re asked to be providing a grade from 1 to 5 and then this formula calculates a grade taking into account the different constraints of the assignment.” (11) For one assignment, the calculated grade is explicitly a **grade suggestion**, which the teacher can choose to copy, but which can also be adjusted: “If you

have two teams that score close together, then it can happen that you do think like ‘well, but this team did slightly better than the other, so they get slightly higher.’ And sometimes it can happen that you’re a bit in between categories. I think that in practice many do just copy the suggestion, but...” (03)

Usage for **feedback** is mixed: two respondents mention that teaching assistants make use of their rubric when giving feedback, though one also notes that “the feedback tends to end up being more about requirements rather than criteria we have in the rubric.” (11) Four others note their assessment guidelines are **not used for feedback**: “the rubrics are not really used at that point. [...] I don’t have the students going through, you know, with a letter in front of them or something.” (02) Both of the courses that apply automated grading also make those tools available so students receive **automated feedback** while working on the assignment.

Four teachers mentioned that **teaching assistants** are the primary audience for and users of their assessment guidelines, either to give feedback or to do (a first pass for) summative grading. This is often combined with **further coordination**, where the assessment guidelines serve as a baseline, but where cases that do not quite fit the mould are discussed by teachers and assistants.

For five courses, the assessment guidelines are **shared with the students**: “of course, the students are aware of the rubric. At least, it has been communicated with them,” (08), though two participants also note that **students don’t always look at it**: “but I think a very large proportion of the students do not really give it much thought,” (08) and “there is a reading comprehension problem that gets worse and worse every year. Half of them cannot even find the rubric.” (09) In three courses, a **modified version is shared** with students, typically with less detail. “The project description explains what criteria are being used and explains based on what they are being

Table 1: Relation between guideline structure and whether the guidelines are shared with students

	shared	modified shared	not shared
rubric	3	1	
list of levels	1	2	
marks		1	2
empty rubric	1		
list of criteria	2		

assessed. And it leaves out the specific levels that these criteria are basically delineated in the rubric.” (11)

In two courses, the rubric is **not shared with students** at all. One such case indicates in the assessment guidelines that the instructor should “remove all pre-written text (lists of common mistakes, good to have)” (05) before sharing the form with the students. Four of the instructors for courses that do not share the rubric, or share a modified version, give the same reason for this choice, namely that they want to prevent **box ticking** behaviour: “[sharing the full rubric] will enable the box ticking behaviour without really promoting the positive aspect of engaging with the assignment, understanding what is being asked, what is necessary for it. [...] It will make it easier, but not necessarily as well done, as rich a work.” (11)

Table 1 shows the relation between the structure of the assessment guidelines (as described in [subsection 5.2](#)) and the degree to which these guidelines are shared with students. Interesting here is that rubrics and plain criteria tend to be shared with the students, whereas lists of levels or marks tend to have only a modified version with less detail shared or not be shared at all. Note that the number of guidelines with marks does not correspond with the number in Figure 2, because in one case we did not discuss whether those marks are shared with students during the interview.

5.4 How do teachers experience their assessment process? (RQ5)

When asked for their experience with the assessment process, our respondents mostly describe challenges which they currently face or which they feel are addressed by some of the assessment strategies they use. The fact that it can be very **time intensive** to assess open-ended assignments was mentioned by seven respondents: “You can do one team per day, so it takes you two weeks full-time to do all. [...] And the week before you’re reading all those peer reviews, so you have a month where you basically don’t do anything else. [...] You want to take everything into account, and you can only do that at the end. That is the big problem.” (06) When we asked for challenges with open-ended assignments in general, as discussed in [subsection 4.1](#), this issue was also brought up by two more respondents, making it nine out of eleven who mentioned it at some point. Solutions come in different forms, for example by employing TAs or using automation: “The structure is really good, because we always have support from the TAs. That’s really helpful, or else, when you have 100 students, it’s really hard to manage.” (02) “We have quite a lot of students, so if we want to assess it, it has to go in some automated way.” (07)

A challenge that results from sharing the work among multiple teachers and TAs is **consistency**. Seven respondents mention that clearly written assessment guidelines help to ensure consistency between multiple assessors: “And it gives clarity. [...] You give consistent instructions and you expect the TAs to actually follow the instructions consistently. To apply them consistently.” (11) For some, the guidelines are the explicit result of collaboration between teachers: “There are [disagreements] several times. There is something that I think ‘ha, this is problematic.’ My colleagues say ‘no, that’s okay.’ And then we try to make an agreement on that. And then when we agree on that, if we think it’s okay or problematic, then we also add it to the evaluation rubric.” (05)

Transparency is another challenge mentioned by five respondents. Respondent 04 mentions that with clear assessment guidelines, “students have a better idea what the assessment ultimately depends upon.” (04) For respondent 07, their automation also helps to ensure transparency: “They can understand where the grade comes from that they get and that it isn’t something we just make up, but that there are clear criteria and those are applied objectively. So I think that is one of the main benefits of automated testing as much as possible or working with detailed rubrics, that it is objective and that there is relatively little discussion.” (07)

Four respondents mentioned that they struggle with the question of assessing **product vs process**: “sometimes it feels like we don’t sufficiently assess the product,” (03) though respondent 09 notes that “the aim of the project is not for them to deliver something to a client. It’s more for them to practice with database APIs, web programming, [...] and for them to learn Agile and Scrum.” (09) For respondent 06 it’s very clear: “This really is the most important aspect of the course: you always assess the process. A good process leads to a good product anyway.” (06)

When students work together, **differentiating assessment in collaboration** can also be a challenge, due to the amount of work and information required to do this. “I think the main challenge is in grading students fairly on their weekly performance and also giving students the impression that they are assessed fairly in that way.” (03) Three respondents also mention **generative AI** as a challenge for assessment. “The main question I have, is how students use gen AI, how we can defend against that or embrace it. I don’t have an answer to that yet.” (08)

Three respondents mention giving **feedback** as a challenge: “When I see what the TAs do, the feedback they give... Well, I’m not always impressed, even though I emphasise it every year.” (08) A challenge mentioned by two respondents is that **tooling is lacking**, for example due to missing features, a mismatch between professional tools and educational goals, fragmentation in tooling or a general lack of quality.

Regarding the assessment guidelines or rubrics specifically, there are both positive and negative experiences. Starting positively, nine respondents said that their process and rubric generally **works well** for them and results in good assessments: “in my experience, it is the case that in the end the average generally matches [the grade] you think the student should receive.” (03) “I had the TAs actually telling me that this actually helped them a lot. And they wish they had it in more courses.” (11) In particular, two respondents mention that their rubric provides a **justification** for the grade: “most of the time it results in the grade we intuitively feel, but we see why and we

can explain it too.” (06) Two other respondents mention that while their rubric may not be perfect, it **covers the common cases**: *“if you can handle 80% of the cases with this, right, then you only have to deal with the 20% that are the complex ones. That’s fine, right?”* (11)

Some respondents also experienced challenges with rubrics, of course. The main complaint, voiced by four respondents, is that rubrics can easily become **over-specified**, turning them into a hindrance rather than a helpful tool. *“My opinion about rubrics is, like, it’s okay, but at some point it’s too much. [...] They’re there to advise you and they’re not really there to check boxes and you know, like, they should be treated that way.”* (02) This is reflected in how this respondent uses their rubric: they have lists of level descriptors, with differing levels per criterion, which are explicitly meant as a tool for holistic grading. Respondent 04 does use a typical rubric with weights for analytic grading, but also mentioned that *“you don’t want to grow the rubric much bigger. [...] I know that it regularly happens that I think, well, that’s not quite covered in there.”* (04) They also mentioned that in their case, it happens that the **outcome does not match their intuition**: *“you often have that you think, ‘oh, I would grant you a higher or lower grade based on what you show here.’ But that can’t be justified in the rubric.”* (04)

Three respondents mentioned that sometimes there is **no matching level**, where none of the descriptors really match the student’s work, in the rubric: *“We have that section with 0, 0.4, 0.7 and 1. We collectively agreed last year that those are discrete measurements. So a 0.6 does not exist. And that often makes it hard, especially [...] if they meet all of them or most of them or the most important, in which box do you go?”* (09)

6 Concluding remarks

Openness in programming assignments is widely celebrated but comes at a cost: across 12 university-level courses, open-ended assignments support realistic application of programming and software engineering knowledge, but make assessment labour-intensive, complex and dependent on judgement distributed among teachers and teaching assistants.

6.1 What is assessed in open-ended programming assignments

Functionality is assessed in every course, even those with self-determined goals, with code quality and design also common. This broadly overlaps with previous work on code assessment: Neutens et al. [21] found functional correctness to be most valued by school teachers, though closely followed by the use of code constructions, which we found to be much less prominent. Stegeman et al. [31] identified aspects such as naming, comments and decomposition in their code quality rubric, which matches some of the code quality aspects mentioned by our respondents. Most courses in our study assessed substantially more than the submitted code (software design, testing, development process, collaboration, version control practices, reflection, reports, presentations), which aligns with common criteria for capstone projects [12]. Within our sample, open-ended programming assessment was mostly an assessment of a broader development activity rather than only of a program as a final product.

This breadth limits the viability of fully automated assessment: automated tests may provide evidence about functionality or code, but outcomes involving design decisions, process, communication and individual understanding continue to require structured human judgement, potentially assisted by tools to organise evidence [28].

6.2 Assessment guidelines as grading and learning resources

Guidelines range from conventional rubrics to lists of levels, marks and plain criteria, sometimes in combination. This goes against a large body of work recommending rubrics [3, 23], but matches findings for capstone assessment [12]. Andrade [2] distinguishes scoring and instructional rubrics, indicating that the clear criteria and descriptions of quality of a good rubric [4] are most valuable for the latter. In the courses we studied, however, rubrics are used primarily to support summative grading, often by teaching assistants. Use for formative feedback is more limited, and detailed versions are not always made available to students.

Our findings suggest that the two purposes impose different demands. Assessor-facing guidelines may need detail for coordination, traceability and justification. A student-facing resource must instead communicate important qualities without degrading to an apparent checklist. Several participants reported withholding or simplifying detailed descriptors to prevent students from focusing on satisfying individual indicators (“box ticking”) rather than on the intended quality of the work. This suggests that assessor calibration, summative grading, formative feedback and student self-regulation might be better served with separate, though aligned, documents.

This distinction may also play into the tension participants described between comprehensiveness and usability. Detailed rubrics can cover common situations and justify a grade, but participants also reported that descriptors may fail to fit unusual submissions and that increasingly detailed rubrics can become cumbersome. Less detailed guidelines leave more room for professional judgement but may offer weaker support for consistency and explanation. The appropriate balance is likely to depend on the assignment, the experience of the assessors and the way exceptional cases are discussed and moderated.

6.3 Consistency, judgement and workload

Participants believed that assessment guidelines improve consistency and make grades easier to justify, especially when teaching assistants perform much of the assessment. This perception is compatible with general literature describing transparency and standardisation as benefits of rubrics [6, 14], but does not demonstrate objective reliability. Studies of programming assessment have found substantial inconsistency among graders even with a rubric [5, 20], and we did not measure inter-rater agreement or independently compare grading decisions.

One possible reading of these findings is that written guidelines are useful but not sufficient. Participants also described discussion among teachers and teaching assistants, adjustment of calculated grade suggestions, and collective decisions about cases that did not fit predefined categories. These practices indicate that assessment consistency may depend on a combination of written criteria, assessor calibration and opportunities for moderation. Retaining some

professional judgement need not mean abandoning consistency; rather, that judgement should ideally be made visible and discussable. It could be that rubrics play a crucial role in facilitating such discussions.

Workload still remains a central concern. Most respondents mentioned the time required for supervision, feedback or grading, and teaching assistants are important to making assessment feasible at such a scale at all. This reinforces the value of support tools, but also constrains what such tools should be expected to accomplish. Based on the practices we observed, potentially useful support includes accommodating different guideline formats, combining analytic and holistic decisions, collecting multiple forms of evidence, coordinating assessors, and recording explanations when an assessor steps away from a calculated suggestion or predefined descriptor. Support for different student-facing and assessor-facing views may also be worth exploring. These are design implications derived from a limited qualitative sample, rather than validated requirements, and should be evaluated in use.

6.4 Limitations

Our findings should be interpreted in light of their limitations. We deliberately selected courses using open-ended assignments, so the results do not describe programming assessment in general. More closed exercises, examinations and courses organised around progressively introduced language constructs may use different assessment practices.

Our study covered 12 courses, mostly at Bachelor level, within computer science programmes at Dutch universities. It was designed to characterise practices rather than estimate their prevalence. The counts reported in the findings show patterns within the sample but do not support claims about how common those practices are in the wider population. Assessment practices in other countries, institutional settings, disciplines or non-CS programming courses may differ. As noted in [subsection 4.1](#), teachers tend to give junior students more structure, and multiple respondents indeed indicated that their Master-level courses are even more open-ended, while being usually less challenging in terms of scale. Another consequence of our focus on Dutch universities could be related to the fact that lecturers there are required to obtain a University Teaching Qualification (UTQ) within a few years of starting at a teaching position [8]. This means that most of the teachers we spoke to, or who designed the course taught by the teacher or teaching assistant we spoke to, have some basic didactic knowledge, including on the topic of assessment. The exact implementation, structure and content of the UTQ differ across universities, however, so with our coverage of seven universities we aimed to ensure that there is enough diversity in their backgrounds. There might be some other cultural consequences as well.

Our account is also based primarily on interviews with one teacher or head teaching assistant per course and on the assessment documents they provided. We did not observe grading sessions ourselves, analyse completed grading forms, measure agreement among assessors or collect student perspectives. Statements about consistency, transparency, checklist behaviour and the usefulness of

rubrics therefore represent participants' experiences and interpretations. They should not be read as independently verified effects of the assessment practices.

Finally, all authors work in an institution where project-based learning and learning-by-doing are prominent. This may have influenced the questions we considered important and our interpretation of open-ended assignments. To limit this influence, we kept the analysis close to the interview data, used second coders on a subset of the material, discussed coding disagreements, and provided supporting quotes throughout the paper alongside codebooks in the supplementary material.

6.5 Future work

Several directions follow from these limitations and findings. To start with an obvious one, studies across countries and institutional contexts could determine which observations are locally specific and which recur more broadly. A larger survey could examine the prevalence of different assessment structures, while retaining clear distinctions among guideline format, grading procedure and audience. A stratification on different types of open-ended assignments could also be of interest in a larger survey.

More direct investigation is also needed into the relationship between assessment guidelines and actual practice. This could include measuring inter-rater agreement, observing how teachers and teaching assistants resolve exceptional cases, and studying how students interpret and use different forms of guidance. In particular, comparisons of shared and audience-specific representations could test the tentative distinction identified here between guidelines designed primarily for learning and those designed primarily for grading.

Finally, our broader project develops tool support for assessing open-ended programming assignments in a slow, evidence-informed fashion. The present study suggests that such support should not assume one rubric format or restrict assessment to source code. We are therefore exploring support for combinations of level descriptors, checklists and open judgements, as well as evidence from code, reports, presentations and other course activities. Future evaluation should examine not only usability and time savings, but also the quality of feedback, assessor agreement, transparency of decisions and the extent to which the tooling preserves appropriate room for professional judgement. Our current prototype supports rubrics, possibly with differing levels per criterion, checklists and open text fields. In an initial pilot of this tool, teaching assistants were positive and indicated they would like to use the tool again and prefer it over the LMS-based rubric we used last year.

Overall, open-ended programming assignments allow students to demonstrate forms of learning that are difficult to capture in closed tasks, but they also make assessment labour-intensive and resistant to complete standardisation. The instructors in this study addressed this through varied assessment guidelines, multiple forms of evidence and a combination of explicit criteria and professional judgement. Understanding how these elements work together and how the needs of students and assessors may differ is an important basis for improving both assessment practice and the tools intended to support it.

References

- [1] Nathalia Da Cruz Alves, Christiane Gresse Von Wangenheim, Jean Carlo Rossa Hauck, and Adriano Ferreti Borgatto. 2020. A Large-scale Evaluation of a Rubric for the Automatic Assessment of Algorithms and Programming Concepts. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM, Portland OR USA, 556–562. doi:10.1145/3328778.3366840
- [2] Heidi Goodrich Andrade. 2005. Teaching With Rubrics: The Good, the Bad, and the Ugly. *College Teaching* 53, 1 (2005), 27–31. doi:10.3200/ctch.53.1.27-31
- [3] John Biggs, Catherine Tang, and Gregor Kennedy. 2022. *Teaching for Quality Learning at University* (5 ed.). Open University Press, McGraw Hill, Maidenhead.
- [4] Susan M. Brookhart and Fei Chen. 2015. The Quality and Effectiveness of Descriptive Rubrics. *Educational Review* 67, 3 (2015), 343–368. doi:10.1080/00131911.2014.929565
- [5] Neil C. C. Brown, Marcus Messer, and Jennifer Ikin. 2025. Failures in Reliably Assessing Program Code Readability. In *Proceedings of the 25th Koli Calling International Conference on Computing Education Research (Koli Calling '25)*. Association for Computing Machinery, New York, NY, USA, 1–7. doi:10.1145/3769994.3770017
- [6] Zenobia Chan and Simone Ho. 2019. Good and Bad Practices in Rubrics: The Perspectives of Students and Educators. *Assessment & Evaluation in Higher Education* 44, 4 (2019), 533–545. doi:10.1080/02602938.2018.1522528
- [7] Phillip Dawson. 2015. Assessment Rubrics: Towards Clearer and More Replicable Design, Research and Practice. *Assessment & Evaluation in Higher Education* 42, 3 (2015), 347–360. doi:10.1080/02602938.2015.1111294
- [8] Johanna de Groot and Rianne Kouwenaar. 2018. *Professionalisation of University Lecturers. The UTQ and Beyond*. Technical Report. VSNU. [Online; accessed 2026-07-16]. <https://www.universiteitenvannederland.nl/files/publications/Professionalisation%20of%20university%20lecturers.pdf>
- [9] Nicole M. Deterding and Mary C. Waters. 2021. Flexible Coding of In-depth Interviews: A Twenty-first-century Approach. *Sociological Methods & Research* 50, 2 (2021), 708–739. doi:10.1177/0049124118799377
- [10] Sue Fitzgerald, Brian Hanks, Raymond Lister, Renee McCauley, and Laurie Murphy. 2013. What Are We Thinking When We Grade Programs?. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 471–476. doi:10.1145/2445196.2445339
- [11] Heidi Goodrich. 1997. Understanding Rubrics. *Educational Leadership* 54, 4 (1997), 14–17. <https://www.ascd.org/el/articles/understanding-rubrics>
- [12] Sara Hooshangi, Asma Shakil, Steve Riddle, Ilknur Aydin, Nayla Nasir, Tejasvi Parupudi, Attiqah Rehman, Michael James Scott, Jan Vahrenhold, Amali Weerasinghe, and Xi Wu. 2026. Evaluating Assessment Practices in Team-Based Computing Capstone Projects. In *Proceedings of the 2025 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR 2025)*. Association for Computing Machinery, New York, NY, USA, 277–312. doi:10.1145/3760545.3783974
- [13] Åse Jevinger and Kristina Von Hausswolff. 2016. Large Programming Task vs Questions-and-Answers Examination in Java Introductory Courses. In *2016 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*. IEEE, Mumbai, 154–161. doi:10.1109/LaTICE.2016.25
- [14] Anders Jonsson. 2014. Rubrics as a Way of Providing Transparency in Assessment. *Assessment & Evaluation in Higher Education* 39, 7 (2014), 840–852. doi:10.1080/02602938.2013.875117
- [15] Maria Kallia. 2017. *Assessment in computer science courses: A literature review*. Technical Report. Royal Society. 1–60 pages. <https://royalsociety.org/-/media/policy/projects/computing-education/assessment-literature-review.pdf>
- [16] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2019. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education* 19, 1 (2019), 1–43. doi:10.1145/3231711
- [17] Joseph S. Krajcik and Namsu Shin. 2014. *Project-Based Learning*. Cambridge University Press, 275–297. doi:10.1017/cbo9781139519526.018
- [18] Collin F. Lynch, Kevin D. Ashley, Vincent Alevan, and Niels Pinkwart. 2006. Defining Ill-Defined Domains; a Literature Survey. In *Intelligent Tutoring Systems (ITS 2006): Workshop on Intelligent Tutoring Systems for Ill-Defined Domains*.
- [19] Angelika Mader, Ansgar Fehnker, and Edwin Dertien. 2020. Tinkering in Informatics as Teaching Method. In *Proceedings of the 12th International Conference on Computer Supported Education*. SCITEPRESS - Science and Technology Publications. doi:10.5220/0009467304500457
- [20] Marcus Messer, Neil Brown, Michael Kölling, and Miaojing Shi. 2025. How Consistent Are Humans When Grading Programming Assignments? *ACM Transactions on Computing Education* 25, 4 (2025). doi:10.1145/3759256
- [21] Tom Neutens, Kris Coolsaet, and Francis Wyffels. 2022. Assessment of Code, Which Aspects Do Teachers Consider and How Are They Valued? *ACM Transactions on Computing Education* 22, 4 (2022), 1–27. doi:10.1145/3517133
- [22] W. James Popham. 1997. What's Wrong—and What's Right—with Rubrics. *Educational Leadership* 55, 2 (1997), 72–75.
- [23] Y. Malini Reddy and Heidi Andrade. 2010. A Review of Rubric Use in Higher Education. *Assessment & Evaluation in Higher Education* 35, 4 (2010), 435–448. doi:10.1080/02602930902862859
- [24] Emma Riese and Olle Bälter. 2022. A Qualitative Study of Experienced Course Coordinators' Perspectives on Assessment in Introductory Programming Courses for Non-CS Majors. *ACM Transactions on Computing Education* 22, 4 (2022), 1–29. doi:10.1145/3517134
- [25] Arthur Rump. 2026. minicoder: Simple in-browser qualitative data analysis tool for plain-text files. doi:10.4121/93041a67-87de-49cc-ad92-2f561ee66852.v1
- [26] Arthur Rump, Angelika Mader, and Vadim Zaytsev. 2026. Supplementary materials for 'How Computer Science Teachers Assess Open-ended Programming Assignments'. doi:10.4121/c1c947eb-279c-49a7-ad7e-56d18d3bdfc9
- [27] Arthur Rump and Vadim Zaytsev. 2022. A Refined Model of Ill-Definedness in Project-Based Learning. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '22)*. ACM, 115–122. doi:10.1145/3550356.3556505
- [28] Arthur Rump, Vadim Zaytsev, and Angelika Mader. 2025. Requirements for an Automated Assessment Tool for Learning Programming by Doing. In *IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 679–686. doi:10.1109/ICST62969.2025.10988998
- [29] D. Royce Sadler. 1989. Formative Assessment and the Design of Instructional Systems. *Instructional Science* 18, 2 (1989), 119–144.
- [30] Sadia Sharmin, Daniel Zingaro, Lisa Zhang, and Clare Brett. 2019. Impact of Open-Ended Assignments on Student Self-Efficacy in CS1. In *Proceedings of the ACM Conference on Global Computing Education (CompEd '19)*. ACM. doi:10.1145/3300115.3309532
- [31] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. 2016. Designing a Rubric for Feedback on Code Quality in Programming Courses. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM. doi:10.1145/2999541.2999555
- [32] Carmen Tomas, Emma Whitt, Rosa Lavelle-Hill, and Katie Severn. 2019. Modeling Holistic Marks With Analytic Rubrics. *Frontiers in Education* 4 (2019). doi:10.3389/feduc.2019.00089
- [33] Dylan Wiliam. 2011. What Is Assessment for Learning? *Studies in Educational Evaluation* 37, 1 (2011), 3–14. doi:10.1016/j.stueduc.2011.03.001