Requirements for an Automated Assessment Tool for Learning Programming by Doing

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

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

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

***Abstract*—Assessment of open-ended assignments such as programming projects is a complex and time-consuming task. When students learn to program, however, they benefit from receiving timely feedback, which requires an assessment of their current work. Our goal is to build a tool that assists in this process by partially automating the assessment of open-ended programming assignments. In this paper we discuss the requirements for this tool, based on interviews with teachers and other relevant stakeholders.**

***Index Terms*—automated assessment, programming education, open-ended assignments, learning-by-doing**

I. Introduction

Learning to program requires feedback, especially formative actionable feedback when it comes to complex issues like style and structure [1]. Formative practices are known to motivate students and strengthen their embrace of the material. However, providing students with feedback can be difficult and time-consuming, especially when teaching large groups. Many tools are available that use unit testing to provide automated feedback on small, closed exercises [2]–[4]. Students will, however, need more open-ended assignments to practice their program design skills [5] when they go beyond the basics of syntax and simple algorithms. Since students need the freedom to make design choices in their projects, every submitted project will be different. In such cases unit tests or comparison with model solutions are no longer a feasible way to automate assessment [6].

Our goal is to develop a tool that supports the assessment process of large open-ended programming assignments. We focus on two programming projects from our university as a case study. In one project, the students are tasked with implementing a multiplayer board game with a client/server architecture. Beyond what was covered in class and the rules of the game, students receive no further guidance or boilerplate code to start from. In the other project, students are even free to choose what they build for the project, as long as they include topics covered in the course. We aim, however, for our tool to be useful in any environment where open assignments are used to teach software design.

Our research question for the project is: *How can a tool support the assessment of open-ended programming assignments?* In this paper, we aim to find out what our stakeholders want such a tool to do: *What requirements do stakeholders have for a tool that supports the assessment of open-ended programming assignments?* To answer this question, we have conducted interviews with our stakeholders, as described in sec. III. The results from these interviews in the form of a list of requirements is described in sec. IV. In sec. V we discuss these results and give some indication of the feasibility of implementing these requirements based on a design and prototype we built. Sec. II introduces some background on assessment, open-ended assignments and automated assessment tools.

II. Background

A. Formative and Summative Assessment

First we need to distinguish between two types of assessment: *formative assessment* and *summative assessment*. Formative assessment has the goal of providing feedback to a student so that they can improve their work, learn and resubmit it [1], whereas summative assessment merely summarises the student's performance, usually expressing it as a grade [7]. Both types of assessment start with an evaluation of the current performance, but for formative feedback the main goal is to provide feedback during the learning, when it is not yet too late to communicate the gap between the current level and reaching the learning goal, and aid in forming strategies to close it [7].

B. Intended Learning Outcomes

The goal we want students to achieve is commonly referred to as the intended learning outcome (ILO). Each ILO is a statement of what a student is expected to have learned at the end of a lecture, a course or study unit or even at the end of a full programme. These statements focus on student behaviour: they describe an action the student should be able to take after finishing the particular unit of study, like "*At the end of this lecture, students can ...*" or "*After following this course, you will be able to ...*" In English, the next parts of these statements will naturally be a verb and some object. The verb is used to describe the action a student should be able to take, and the object describes the relevant course topic. The object and verb cover two aspects of learning: the content that should be learned, as well as the level of understanding that is desired. The latter is often categorised through a taxonomy like Bloom's

979-8-3315-0814-2/25 © 2025 IEEE

ICST 2025, Naples, Italy
Education Track

taxonomy [8] or the SOLO taxonomy [9], which link certain verbs with a corresponding level of cognitive processing.

Including a verb in an ILO not only indicates the level of understanding students are expected to acquire of that content, but also helps to design learning activities and assessment tasks that actually teach students how to perform that activity. If our intention is for students to learn how to design and develop software of average size (10–20 classes), then students should actually spend time designing and developing software during the course. If the only learning activities are, for instance, lectures on the principles of object-oriented programming, it would not be surprising if students do not meet the intended outcome of being able to develop software by themselves. Similarly, a written exam is not well-suited to assess this ability, so a task should be chosen where students can actually demonstrate their ability to develop software. This principle is called *constructive alignment*: learning activities, assessment tasks, and ILOs — all three should be aligned within each course [10].

C. Open-ended Assignments

Following the principles of constructive alignment, a student's performance in program design is best assessed in a larger task where they actually have to make design choices. A constructivist view of education also argues that these open-ended assignments are necessary for learning. An open question or problem is the starting point for learning in forms of learning with a base in this tradition, like the *driving question* in project-based learning [11], [12] or *wicked problems* in challenge-based learning [13], [14].

Open-ended projects are different from typical assignments assessed by automated tools, because there exists no single definite solution to which submissions can be compared. Open problems that do not have a definite solution are sometimes called *ill-defined* — the indefinite endpoint is one of three criteria given by Simon [15] for calling a problem ill-defined. Ill-definedness is not a binary classification, however, but a continuum, ranging from well-defined to ill-defined [16]. Later work describes the space of ill-definedness along two axes: the number of alternative solution strategies and implementation *variability* within a strategy, and the objective solution *verifiability* [17]. We have recently extended the verifiability axis and used the model to analyse two programming projects [6]. Rather than trying to classify a project as well- or ill-defined as a whole, we considered the assessment criteria individually. We found that even though students can get a lot of freedom to make decisions in an open-ended assignment, many criteria are on the well-defined side of the spectrum, making automated assessment of these criteria feasible.

D. Automated Assessment Tools

Automated assessment tools for programming assignments have existed for a long time and use a variety of dynamic and static techniques, but most are designed for small, well-defined exercises that can be evaluated with input/output tests or by comparing to model solutions provided by teachers [2], [3], [18], [19]. Configuring these tools to support new exercises is often difficult or even impossible [20] and many are known disincentivize creative or innovative solutions [21].

More recently, tools have started using data-driven techniques [22]–[24], which seems a more promising approach for open-ended assignments. By analysing patterns in historical student submissions, these tools can learn to recognize valid solutions without requiring teachers to enumerate all possibilities. By leveraging the natural variation present in student submissions, such tools can cover variation in approaches without extra configuration, as would be required for most other techniques. The only exception is automated testing, which does not care about implementation variation at all, but that approach is limited to only objectively assessable assignments with strictly defined interfaces.

There are several approaches to using student data: clustering techniques, for example, identify groups of similar solutions [25]–[29], allowing teachers to efficiently provide feedback at the cluster level. While this is helpful in scaling manual feedback, it does not support teachers in the actual assessment of how a student's work meets the criteria. Other tools extract common code patterns across submissions to highlight patterns that might prove of interest [30], [31] or use these patterns as features to train machine learning models to identify positive and negative examples [32], [33]. Extracting these patterns requires careful configuration for each dataset to get the best results [31], however, which makes it difficult to reuse across assignments and especially tricky for larger open-ended assignments. Some data-driven tools interactively involve teachers in the learning phase, for example to mark which patterns indicate correct and incorrect solutions [34], to guide the system in finding equivalent approaches [35] or to define rules based on model solutions [36], [37], but they are still designed for assignments where students' solutions mostly vary by implementation rather than design, as is the case in open-ended assignments.

Existing tools that work on assignments that are not objectively verifiable, typically depend on heuristic techniques that only consider some aspects, rather than the full solution [17]. Nye at al [38] recommend that ill-defined domains are split into well- and ill-defined parts, such that the well-defined parts can be handled by a tool.

Recent advances in generative AI lead many to reconsider how we will teach programming in the future and also how we will assess students' programming skills [39]. Experiments with GPT-3.5 showed that it often failed to provide accurate feedback on programming assignments, even on relatively small exercises [40]. GPT-4 improved these results, and on small exercises its generated feedback can lead students to better results than human-written feedback [41], but still provides incomplete or incorrect feedback in almost half of the cases [42]. Even when these models improve and are able to give more accurate feedback, the lack of transparency in their workings will remain an issue when using them directly in the assessment process.

Overall, we see that although there are many tools for

Table I: Number of participants per type of stakeholder. Note that many participants represented more than one stakeholder.

Role	#Participants
Teaching staff: coordinators	4
Teaching staff: configurators	4
Teaching staff: tutors in detail	7
Teaching staff: tutors in overview	4
Students	3
Examination board	1
Programme management	0
Educational support	4
Developer/maintainers	0
System administrators	0

automated assessment of small, closed programming exercises, there is limited work on open-ended assignments. This brings us back to the question we answer in this paper: what requirements do stakeholders have for a tool that supports the assessment of open-ended programming assignments?

III. Interviews Setup

To answer this question, we held interviews with stakeholders of our intended tool[1]. We had three goals with these interviews, namely to identify our list of stakeholders, to identify their goals and applications for the tool, and to identify requirements related to those goals and applications.

We selected participants based on their role and involvement with open-ended assignments at our university. We started with teaching staff and iteratively determined stakeholders by asking each participant who else would be relevant to talk to. In total, we invited 19 stakeholders to an interview, 12 of whom agreed and participated in the study. Many participants represented multiple stakeholders based on their different roles, for example teaching assistants who are themselves students in other courses and teachers who would interact with the tool in different roles. The distribution of participants over different roles is shown in tbl. I. These roles correspond to our types of stakeholders, which we will describe in sec. IV.

Not all stakeholders were represented in the interviews. We based the concerns of these stakeholders on their role description, general concerns and how those could relate to the project. The *programme management's* concerns were brought in by one of the authors, who is a programme director. The concerns of *developers/maintainers* were based on another author's previous experience in developing a tool with similar goals. As we will be focussing on a prototype first, we decided to not investigate the concerns of *system administrators* further either, as those would likely be at the edge of our system and not important in the scope of a prototype.

In the interviews, we asked participants about their use cases for an automated assessment tool and their concerns related to those use cases. We started each interview with an introduction, briefing and handling informed consent. After this introduction, we asked questions about what the participant would like an automated assessment tool to do or not do from their different perspectives (e.g. as a teaching assistant or as a student). Their concerns were summarised and noted down during the interview. After discussing the concerns from each perspective, the participants were asked to review the concerns that were recorded in the document and make any corrections or clarifications. The interview concluded with a debriefing about the next steps and how their responses would be used.

The results of these interviews were lists of concerns for each stakeholder.[2] Given that we created this artefact during each interview, we decided not to record the interviews. To determine the requirements for our tool, we combined all similar concerns across our stakeholders and noted the number of participants that voiced a certain concern to serve as a weak indicator of importance. One examination board member noted, for example, that the tool "should not be 100% automatic" and a coordinator expressed their concern that the "teacher gets final say in the grade," which we both included in the requirement that the tool "does not give an (automatic) grade." Next, we distinguished functional requirements and quality requirements, resulting in a final list of requirements, which we describe in sec. IV. From the functional requirements we distilled a set of use cases, which serve as a categorisation to better get an overview of the requirements and at the same time justifies their applicability to the tool.

Though not the focus of this study, we did create a design and prototype based on that design to test the feasibility of implementing our requirements. To evaluate our design, we used a scenario-based method [43], [44] to evaluate whether the design supports different usage scenarios, which we derived from the use cases mentioned before. We also performed a small-scale evaluation of our prototype with the assessment criteria and student submissions of two programming projects.

IV. Results

From our initial list of stakeholders and the recommendations from our interview participants, we found the following ten stakeholders for our project:

- **Teaching staff**, both teachers and teaching assistants. They have different relations to the tool depending on their role in a course: *coordinators* may choose to use the tool in a course, *configurators* set the tool up with the appropriate assessment configuration, *tutors in detail* give feedback to students on an individual level or grade a single submission, and *tutors in overview* want to incorporate generic feedback in plenary teaching.
- **Students** have an interest in receiving feedback on their work.
- The **examination board** is responsible for the quality of assessment and grading.
- **Programme management** is responsible for the study programme as a whole and the quality of content.

[1]These interviews were conducted according to the outlined procedure with approval from the Ethics Committee Computer & Information Science of the University of Twente. This is known to them as request *RP 2022-178*.

[2]The templates used to gather concerns are available in the companion package at https://doi.org/10.5281/zenodo.14627201.

- **Educational support** helps educators in the organisation of their teaching, including choosing the tools they may want to use.
- **Developers and maintainers** are, of course, responsible for building the tool and adapting it to a changing environment.
- **System administrators** are responsible for deploying the tool and keeping it running and up-to-date.

We gathered the concerns for all of these, except for system administrators, and determined their use cases and associated requirements from those. In the following sections we describe the requirements as related to their use cases. The full list is available online[3]. In the next sections, we refer back to the requirements in this list with their identifiers in parentheses after the relevant discussion. Identifiers starting with *F* indicate a functional requirement, identifiers starting with *Q* a quality requirement.

A. Requirements for Individual Assessment

We started the interview process with two use cases in mind for our tool: *providing individual feedback to students* (formative assessment) and *grading a project* (summative assessment). Note that these are both use cases for teaching staff, be it lecturers or teaching assistants, because we only focus on the first part of feedback: assessing the current state. Giving actionable feedback also requires identifying what needs to be improved and providing suggestions on how to achieve that, which we trust the teaching staff to do.

When it comes to these use cases, the three requirements that were mentioned most often were that the tool should *not* give an (automatic) grade (F12), that the tool should give a quick and clear overview of the assessment (F3) and that the tool should show a student's progress over time (F5). These were all mentioned by tutors in detail, with the first one also supported by coordinators and the examination board. Based on the interviews, we split the use case of grading a project into two subtly different use cases: *grading a project in a grading session* and *grading a project during an oral exam*. For the first one, tutors indicated an interest in reviewing projects "horizontally", i.e. working through projects per criterion, rather than reviewing all criteria project by project (F13). For the latter use case, this is not helpful, but tutors did mention their interest in a student's progress over time (F5).

Related to giving an overview, tutors in detail also mentioned wanting to get suggestions for interesting code snippets to look into (F1), see code snippets as evidence for an assessment (F6) and get more context and details for those snippets (F26). They would also like to see similar problems and repetitions of the same problem grouped together (F2) and have the option to ignore all similar mistakes (F4). Additionally, the tool should help them understand and navigate the program (F23).

When providing individual feedback, tutors mentioned they would like to share feedback from the tool with students (F10), extending or modifying it to make it more constructive (F8). The standard messages from the tool should use objective language for objective observations (F24). Coordinators and the examination board also expressed an interest in teachers being able to make their own assessment (F11). Tutors also mentioned the tool should never share feedback with students automatically (F9). One student had a similar concern, wanting all feedback to be checked by a teacher or teaching assistant before it was shown to them (F59).

B. Requirements of Students

Of course, students also have two use cases for the system: *receiving feedback* and *getting a grade*. While the latter is likely best served through an integration with existing platforms that students already interact with (F28), both students and tutors in detail indicated they would like the tool to help students understand feedback, for example by linking to explanations of terminology (F22). Additionally, students indicated they would like to see their own progress over time (F5), receive clear (F68) and constructive (Q18) feedback, also on things that are going well (F67). They would also like the assessment process to be understandable (Q16) and see a marked rubric, especially when a project is graded summatively (F70). One coordinator agrees, noting that the tool should be able to explain the given assessment on demand (F27). In the context of formative feedback, students would like to see which tutor has shared feedback with them (F69).

C. Requirements for Aggregate Assessment and Evaluation

Besides individual assessments, teachers also indicated an interest in *providing more general feedback to the whole cohort during plenary sessions* and *giving help to groups of students with similar needs*. For the first use case, tutors in overview indicated they would like to see what criteria or ILOs are commonly met or not met (F25) and see the progress of the cohort as a whole during the course (F14). For the second use case, one tutor in overview mentioned they would like to see clusters of students who make similar mistakes (F15). One coordinator noted that the information shown should not be overwhelming when there are 300 projects submitted (Q8).

An additional use case that came up was to *evaluate a course*, of interest to teaching staff, but also to programme management and educational support. A part of this is analysing the outcomes, which is also relevant to the use cases on providing feedback to groups of students. Tutors in overview indicated they would like to see clusters of mistakes that happen together (F16), see code samples (F17) and normalised patterns (F18) for common mistakes in a group and compare metrics (F19). The horizontal review mentioned for the grading process (F13) could also be part of this outcome analysis. Comparing the outcomes of different groups based on meta-information, like gender or prior knowledge, (F20) and comparing the outcomes with previous editions of a course (F21) were also mentioned as part of the evaluation use case. One tutor in overview mentioned they would like to see a dashboard or report with this evaluation at the end of the course (F57). Additionally,

[3]The full list of requirements is available in the companion package at https://doi.org/10.5281/zenodo.14627201.

the examination board and educational support indicated an interest in using the tool to evaluate the constructive alignment between ILOs and assessment criteria (F58).

D. Requirements for Configuration

To support all these use cases, the tool needs to be configured by teaching staff. This is not a use case per se, but it is necessary to enable the other use cases. The two most-mentioned requirements in this category were that the configurators would like to have a community or platform in which they can share ILOs, criteria and rules (F49) and that the tool gives support while configuring ILOs, criteria and rules, for example with documentation, examples and a live assessment of an example project (F54). To further support configuration the tool should give suggestions for rules (F43) and discover patterns that can be matched with criteria (F53).

Regarding ILOs and constructive alignment between ILOs and criteria, the tool can support configurators by giving guidance in formulating the ILOs (F32), linking criteria to one or more ILOs (F51) (but also allow criteria that do not link to an ILO (F50)) and getting feedback on the ILOs (F55) and the link between ILOs and criteria (F56). To keep the assessment configuration organised, the tool should support grouping criteria together (F44) and support configuring the importance of different criteria (F52). Within the configuration UI, options should be sensibly grouped together (Q4) and unused options should be hidden (Q5).

Many other requirements mentioned different aspects of a program that the tool should support assessment of, like code patterns and design patterns (F38), comments and code file headers (F42), variable naming conventions (F33), use of constructs like if-statements (F34), metrics from static analysis like length of a method (F35), code style consistency (F36), metrics from dynamic analysis like test coverage or execution time (F37), code smells (F39), code quality (F40) and unused code (F41). Besides this flexibility in supported criteria, configurators were also interested in flexibility regarding the course structure, like changing the focus to different ILOs during consecutive parts of a course (F45) and only seeing those ILOs and criteria that are relevant at that point in the course (F46). Across courses, a configurator noted they would like to copy relevant parts from previous course editions (F48), separate parts of criteria that change every edition from ones that stay the same (F29) and for the tool to support assessment of overarching ILOs that come back in many courses (F7).

Coordinators also had concerns about the configuration, like the tool not imposing restrictions on the types of exercises given (F31), working with exercises where students have to improve a given piece of code (F47) and the tool being able to assess non-functional aspects (F30). They would also like to easily import and export rubrics (Q7) and they think the tool should provide basic functionality with minimal configuration (Q6) to quickly get started. Of course the tool should fit in the context of the course (F60) and support the programming language used in the course (F61). The examination board also noted that criteria are not always binary, and the tool should support those criteria too (F62).

The flexibility that is required here also leads to two requirements from the developer's side: adding support for new programming languages (Q1) as well as new metrics or types of patterns or rules (Q2) should be easy to achieve.

E. Requirements for Interoperability

The second most mentioned requirement, after not automatically giving a grade, was that our tool should integrate with existing platforms (F28). Coordinators as well as tutors in detail mentioned that they would like the tool to integrate in systems they already use, like the Canvas learning management system (LMS) or GitHub. One coordinator added that these integrations should be seamless and built with attention to details (Q3).

F. General Requirements

Our stakeholders also expressed some concerns about the functioning of the tool in general. Two important ones to note concern reliability and validity: in summative contexts, the assessment should be highly reliable and valid (Q9), while in formative contexts, the reliability and validity should be high enough to serve as useful discussion starter (Q10). Other stakeholders expressed that the assessment should be reliable enough to be trusted (Q14) and in general reliable, valid and transparent (Q15).

Besides the assessment, the tool itself should also work and "not spew error messages" (Q11), have fast and easy interactions and quick navigation (Q17) and be easy, simple and intuitive to use, especially on the first interaction (Q13). One coordinator also expressed a desire to tweak every little setting to fully customise the tool to their preferences (Q12).

G. Requirements Out of Scope

Not all requirements voiced by our participants could be connected to a use case for our tool, so we deemed four requirements to be out of scope for this project. Coordinators, tutors in detail and the examination board requested that the tool includes or works with a plagiarism checker (F63), but we consider this to fall out of scope, because our tool was already requested to integrate with other platforms—cf. (F28). These often already come with tools for plagiarism checking, integrated or built in directly. The examination board also requested that the tool helps in the inspection of borderline cases when it comes to grades (F64). This is also out of scope, however, because our tool does not perform automatic grading and thus has no ability to highlight borderline cases automatically. Finally, a tutor in overview noted they would like the tool to analyse the consistency between different assessors overall (F65) or on a single project (F66), but this would require comparing manually entered assessments, which is beyond the scope of the project.

V. Discussion and Conclusion

Looking at the results, we recognise two major themes coming back in many requirements: our tool should be *supporting* and *flexible*. It should be *flexible*, because it should support every criterion that could be assessed by looking at code and every such criterion in a course should be configurable in the tool. It should integrate with LMSs and other tools and be extensible to support multiple programming languages. It should be *supporting*, because it should help teachers assess projects by highlighting the code that is relevant to certain assessment criteria, rather than performing automatic grading by itself. The tool should also leverage previous submissions to support teachers during the configuration phase with suggestions and immediate feedback.

A. Limitations

As always when doing interviews, there is a risk of misinterpretation. In our case, we need to interpret what our participants have said and ultimately translate it into our list of requirements. We mitigated this risk by taking notes and repeating back what was written, guiding our participants into formulating concerns directly. Our participants were able to review their documented list of concerns and make corrections if there were any misunderstandings during the interview. By doing the first step of analysis with the participant present, we could ensure that there was no loss of meaning or confusion introduced by this step.

Another limitation stems from our pool of participants: the sample is small and all of them had experience working at a single institution. This means that we were limited in the diversity of perspectives we were able to get. On one hand, this is a good feature, since this is the primary environment we are designing for. On the other hand, focusing on the university with a learning-by-doing vision [45] and a history of large integrated study units with integrating projects [46], can be a source of certain biases. We were able to mitigate some aspects of this by generalising requirements that were formulated in terms of our environment—for example, by generalising concerns about integrating with our LMS to integrating with LMSs in general.

In the process of conducting our interviews, we have discovered two more issues. First, we used the classic CSAT/DSAT method of measuring customer satisfaction, by asking participants to prioritise their concerns with scoring how satisfied they would be if the tool addressed their concern, and how dissatisfied they would be if it did not. During the interviews we noticed our participants were confused about this system. Some participants seemed to interpret high scores as high dissatisfaction while others interpreted low scores as being highly dissatisfied. Clarification during the interviews did not help and only led to internally inconsistent results. This means that these scores were not usable and had to be discarded. In the end we only used the number of participants who voiced a concern as a weak indicator of priority.

Second, we noticed that some participants had difficulty separating their different roles, despite repeated guidance in the formulation of our questions. This means that a concern like "the tool supports Python and GitHub" was recorded as a concern for a *tutor in detail* even though it is not their decision to use the tool in a course or not. This could have been an issue if we had to prioritise concerns with the power-interest classification of stakeholders who voiced those concerns, but we did not find any conflicting concerns in the process of translating them to requirements, so this was not necessary.

B. Practical Evaluation and Future Work

As noted in sec. III, we have created a design and developed a prototype to test the feasibility of implementing these requirements. Since the main themes of support and flexibility are mainly concerned with the main features of the assessment process and the types of criteria it supports, we decided to focus on the core assessment pipeline. In a scenario-based evaluation, we found that our design is able to support all use cases and fulfils many requirements. Out of the 89 total requirements, our design did not explicitly address ten, four of which we already classified as out of scope. The other six were mentioned only by a single stakeholder during the interviews.

Of these six, only two requirements about supporting rubrics (F70, Q7) are clearly not met, because we focus on providing feedback rather than calculating grades. It is therefore more important to get feedback on each specific criterion, rather than seeing what score you would receive according to a rubric. The other four are not explicitly addressed, but there are no design choices that prevent these from being addressed. We, for example, did not explicitly address the option to separate criteria that change between editions from criteria that stay the same (F29), but users can achieve this using the standard categorisation features for criteria.

Finally, we built a prototype of the core of our design, to test the feasibility in practice. We configured the prototype with the ILOs and assessment criteria of our two case study projects and compared the resulting automated assessments of student projects with the corresponding manual assessments. While we were able to implement the prototype without issues, the tool was not yet at the level where we want it to be regarding *flexibility* and *support*. There were still some criteria that did not fit in the model that our design proposed, even though they were (partly) automatically assessable in principle, so the model was not quite flexible enough. The features that should support teachers in the configuration phase were not helpful in practice. The configuration process is still a major area where improvements can be made, to actually support teachers in using the tool.

C. Summary of Findings

In our efforts to build an automated assessment tool that supports teachers in the assessment of open-ended programming assignments, we conducted interviews with teaching staff and other relevant stakeholders to identify their goals, applications and related requirements for such a tool, and reported on them qualitatively. Two major themes emerge from these requirements, regarding *flexibility* (being able to assess a wide variety of criteria, integrating with other platforms

and tools and being extensible to multiple programming languages) and its *supporting* role (by aiding teaching staff in the assessment process, rather than fully automating, but also guiding them in the configuration of the tool). To some extent this contradicts the existing trend in highly specialised nanotutors and autograders.

It remains an open question how to fully address all these requirements in a reliable tool. We were able to address most of the most prominent ones in our design. However, the prototype of the core of it revealed challenges in both flexibility and support. In its current state, the prototype does not yet support the entire spectrum of widely varying assessment criteria used in practice, nor does it advise the teachers in its own configuration. More work will follow to address these issues.

References

[1] R. Kneyber, D. Sluijsmans, V. Devid, and B. W. Lopez, *Formative Action: from Instrument to Design*. Hodder Education Group, Apr. 2024.

[2] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83–102, Jun. 2005. [Online]. Available: https://doi.org/10.1080/08993400500150747

[3] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research - Koli Calling '10*. ACM Press, 2010. [Online]. Available: https://doi.org/10.1145/1930464.1930480

[4] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Automated grading and feedback tools for programming education: A systematic review," 2023. [Online]. Available: https://doi.org/10.48550/ARXIV.2306.11722

[5] A. Mader, A. Fehnker, and E. Dertien, "Tinkering in informatics as teaching method," in *Proceedings of the 12th International Conference on Computer Supported Education*. SCITEPRESS - Science and Technology Publications, 2020. [Online]. Available: https://doi.org/10.5220/0009467304500457

[6] A. Rump and V. Zaytsev, "A refined model of ill-definedness in project-based learning," in *MoDELS Companion Proceedings*. ACM, oct 2022. [Online]. Available: https://doi.org/10.1145/3550356.3556505

[7] D. R. Sadler, "Formative assessment and the design of instructional systems," *Instructional Science*, vol. 18, no. 2, pp. 119–144, Jun. 1989. [Online]. Available: https://doi.org/10.1007/bf00117714

[8] D. R. Krathwohl, "A revision of bloom's taxonomy: An overview," *Theory Into Practice*, vol. 41, no. 4, pp. 212–218, Nov. 2002. [Online]. Available: https://doi.org/10.1207/s15430421tip4104_2

[9] J. Biggs and K. Collis, *Evaluating the Quality of Learning: The SOLO Taxonomy*. Academic Press, 1982.

[10] J. Biggs and C. Tang, *Teaching for quality learning at university*, 4th ed. McGraw-Hill/Society for Research into Higher Education/Open University Press, 2011.

[11] P. C. Blumenfeld, E. Soloway, R. W. Marx, J. S. Krajcik, M. Guzdial, and A. Palincsar, "Motivating project-based learning: Sustaining the doing, supporting the learning," *Educational Psychologist*, vol. 26, no. 3-4, pp. 369–398, Jun. 1991. [Online]. Available: https://doi.org/10.1080/00461520.1991.9653139

[12] J. S. Krajcik and N. Shin, "Project-based learning," in *The Cambridge Handbook of the Learning Sciences*, R. K. Sawyer, Ed. Cambridge University Press, 2014, pp. 275–297. [Online]. Available: https://doi.org/10.1017/cbo9781139519526.018

[13] J. Lönngren, "Wicked problems in engineering education: Preparing future engineers to work for sustainability," Ph.D. dissertation, Chalmers University of Technology, Gothenburg, Sweden, 2017.

[14] S. E. Gallagher and T. Savage, "Challenge-based learning in higher education: an exploratory literature review," *Teaching in Higher Education*, pp. 1–23, Dec. 2020. [Online]. Available: https://doi.org/10.1080/13562517.2020.1863354

[15] H. A. Simon, "Information-processing theory of human problem solving," in *Handbook of Learning and Cognitive Processes (Volume 5)*, W. Estes, Ed. Psychology Press, 1978.

[16] P. Fournier-Viger, R. Nkambou, and E. M. Nguifo, "Building intelligent tutoring systems for ill-defined domains," in *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2010, pp. 81–101.

[17] N.-T. Le, F. Loll, and N. Pinkwart, "Operationalizing the continuum between well-defined and ill-defined problems for educational technology," *IEEE Transactions on Learning Technologies*, vol. 6, no. 3, pp. 258–270, Jul. 2013. [Online]. Available: https://doi.org/10.1109/tlt.2013.16

[18] N.-T. Le and N. Pinkwart, "Towards a classification for programming exercises," in *Proceedings of the 2nd Workshop on AI-supported Education for Computer Science*, Jan. 2014.

[19] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Automated grading and feedback tools for programming education: A systematic review," *ACM Transactions on Computing Education*, vol. 24, no. 1, pp. 1–43, Feb. 2024. [Online]. Available: https://doi.org/10.1145/3636515

[20] H. Keuning, J. Jeuring, and B. Heeren, "A systematic literature review of automated feedback generation for programming exercises," *ACM Transactions on Computing Education*, vol. 19, no. 1, pp. 1–43, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3231711

[21] M. G. Hahn, S. M. B. Navarro, L. De La Fuente Valentin, and D. Burgos, "A systematic review of the effects of automatic scoring and automatic feedback in educational settings," *IEEE Access*, vol. 9, pp. 108 190–108 198, 2021. [Online]. Available: https://doi.org/10.1109/access.2021.3100890

[22] J. C. Paiva, J. P. Leal, and Á. Figueira, "Automated assessment in computer science education: A state-of-the-art review," *ACM Transactions on Computing Education*, Feb. 2022. [Online]. Available: https://doi.org/10.1145/3513140

[23] J. McBroom, I. Koprinska, and K. Yacef, "A survey of automated programming hint generation: The HINTS framework," *ACM Computing Surveys*, vol. 54, no. 8, pp. 1–27, Nov. 2022. [Online]. Available: https://doi.org/10.1145/3469885

[24] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Machine learning-based automated grading and feedback tools for programming: A meta-analysis," in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ACM, Jun. 2023. [Online]. Available: https://doi.org/10.1145/3587102.3588822

[25] S. Gross, X. Zhu, B. Hammer, and N. Pinkwart, "Cluster based feedback provision strategies in intelligent tutoring systems," in *Intelligent Tutoring Systems*. Springer Berlin Heidelberg, 2012, pp. 699–700.

[26] E. L. Glassman, J. Scott, R. Singh, P. J. Guo, and R. C. Miller, "OverCode," *ACM Transactions on Computer-Human Interaction*, vol. 22, no. 2, pp. 1–35, Apr. 2015. [Online]. Available: https://doi.org/10.1145/2699751

[27] J. B. Moghadam, R. R. Choudhury, H. Yin, and A. Fox, "AutoStyle," in *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*. ACM, mar 2015. [Online]. Available: https://doi.org/10.1145/2724660.2728672

[28] R. R. Choudhury, H. Yin, and A. Fox, "Scale-driven automatic hint generation for coding style," in *Intelligent Tutoring Systems*. Springer International Publishing, 2016, pp. 122–132.

[29] A. G. Zhang, Y. Chen, and S. Oney, "VizProg: Identifying misunderstandings by visualizing students' coding progress," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2023. [Online]. Available: https://doi.org/10.1145/3544548.3581516

[30] A. Nguyen, C. Piech, J. Huang, and L. Guibas, "Codewebs: scalable homework search for massive open online programming courses," in *Proceedings of the 23rd international conference on World wide web - WWW '14*. ACM Press, 2014. [Online]. Available: https://doi.org/10.1145/2566486.2568023

[31] K. Mens, S. Nijssen, and H.-S. Pham, "The good, the bad, and the ugly: mining for patterns in student source code," in *Proceedings of the 3rd International Workshop on Education through Advanced Software Engineering and Artificial Intelligence*. ACM, Aug. 2021. [Online]. Available: https://doi.org/10.1145/3472673.3473958

[32] T. Lazar, M. Možina, and I. Bratko, "Automatic extraction of AST patterns for debugging student programs," in *LNCS*. Springer International Publishing, 2017, pp. 162–174.

[33] M. Možina and T. Lazar, "Syntax-based analysis of programming concepts in python," in *LNCS*. Springer International Publishing, 2018, pp. 236–240.

[34] M. Možina, T. Lazar, and I. Bratko, "Identifying typical approaches and errors in prolog programming with argument-based machine learning," *Expert Systems with Applications*, vol. 112, pp. 110–124, Dec. 2018. [Online]. Available: https://doi.org/10.1016/j.eswa.2018.06.029

[35] S. Xu and Y. S. Chee, "Transformation-based diagnosis of student programs for programming tutoring systems," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 360–384, apr 2003. [Online]. Available: https://doi.org/10.1109/tse.2003.1191799

[36] A. Mitrovic, "Fifteen years of constraint-based tutors: what we have achieved and where we are going," *User Modeling and User-Adapted Interaction*, vol. 22, no. 1-2, pp. 39–72, Oct. 2011.

[37] P. Suraweera, A. Mitrovic, and B. Martin, "A knowledge acquisition system for constraint-based intelligent tutoring systems," in *Proceedings of the 12th International Conference on Artificial Intelligence in Education (AIED)*, ser. Frontiers in Artificial Intelligence and Applications, C. Looi, G. I. McCalla, B. Bredeweg, and J. Breuker, Eds., vol. 125. IOS Press, 2005, pp. 638–645.

[38] B. D. Nye, M. W. Boyce, and R. A. Sottilare, *Defining the Ill-Defined: From Abstract Principles to Applied Pedagogy*. Army Research Laboratory, 2016, pp. 19–37.

[39] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming is hard - or at least it used to be," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. ACM, Mar. 2023. [Online]. Available: https://doi.org/10.1145/3545945.3569759

[40] R. Balse, B. Valaboju, S. Singhal, J. M. Warriem, and P. Prasad, "Investigating the potential of GPT-3 in providing feedback for programming assessments," in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ACM, Jun. 2023. [Online]. Available: https://doi.org/10.1145/3587102.3588852

[41] H. Nguyen, N. Stott, and V. Allan, "Comparing feedback from large language models and instructors: Teaching computer science at scale," in *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, ser. L@S '24. ACM, Jul. 2024, pp. 335–339. [Online]. Available: https://doi.org/10.1145/3657604.3664660

[42] I. Azaiz, N. Kiesler, and S. Strickroth, "Feedback-generation for programming exercises with gpt-4," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE 2024. ACM, Jul. 2024, pp. 31–37. [Online]. Available: https://doi.org/10.1145/3649217.3653594

[43] L. Dobrica and E. Niemela, "A survey on software architecture analysis methods," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 638–653, jul 2002. [Online]. Available: https://doi.org/10.1109/tse.2002.1019479

[44] A. Patidar and U. Suman, "A survey on software architecture evaluation methods," in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2015, pp. 967–972. [Online]. Available: https://ieeexplore.ieee.org/document/7100391

[45] University of Twente, "Learning-by-Interacting: The University of Twente Vision on Learning and Teaching," https://www.utwente.nl/en/service-portal/organisation-regulations-and-codes-of-conduct/vision-on-learning-and-teaching, Apr. 2023.

[46] I. Visscher-Voerman and A. Muller, "Curriculum Development in Engineering Education: Evaluation and Results of the Twente Education Model (TOM)," in *45th SEFI Annual Conference*, 2017. [Online]. Available: https://ris.utwente.nl/ws/portalfiles/portal/19806823/sefi_curriculum_development_in_engineering_education.pdf