Summative Assessment of Program Comprehension in CS1 with Questions About Learners' Code

Afonso B. Caniço
ISTAR, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

André L. Santos
ISTAR, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Abstract

Questions about Learners' Code (QLCs) assess to what extent learners understand their own code through personalized questions that target previously written code. We conducted an experiment involving 18 students of an Introductory Programming (CS1) course taught in Java, comparing the scores of their midterm test based on code-writing questions with the scores of a multiple-choice test based on QLCs. The questions targeted student code submitted throughout the semester in an automated assessment system. Regression analysis shows a strong association between the scores of both tests, suggesting comparable discriminatory power. Participants generally reacted positively and recognised the benefits of this kind of assessment. Our results suggest that QLC tests may serve as a valid complement for CS1 summative assessment targeting program comprehension.

2012 ACM Subject Classification Social and professional topics → Software engineering education; Social and professional topics → Student assessment

Keywords and phrases introductory programming, CS1, programming education, student assessment, program comprehension, questions about learners' code

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.1

Supplementary Material *Software (Source Code)*: `https://github.com/ambco-iscte/jask` [1]

Funding This research was partially supported by national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT), ISTAR-Iscte Pluriannual Projects UID/04466/2025 (`https://doi.org/10.54499/UID/04466/2025`), and an ISCTE-IUL scholarship.

Acknowledgements We thank the anonymous participants for taking part in the study.

1 Introduction

Traditionally, introductory programming students are tasked with assignments focusing on writing programs that produce specific outputs. However, students hold misconceptions [26], and struggle to explain their own code even when they have successfully completed an assignment [13], indicating that completion does not necessarily imply comprehension.

Given that a learner's ability to produce working code does not imply full understanding of the underlying concepts, student efficacy is affected, as programming students tend to perform better when they work towards mastering assignments' underlying concepts [33]. Code-reading skills are an important prerequisite for problem-solving [19].

Students may overlook the importance of not only producing a working program, but also understanding its underlying concepts [16]. Moreover, the emergence of Large Language Models (LLMs) – capable of generating working solutions to most introductory programming problems [6, 7, 29] – introduces a new paradigm for students, who may overrely on LLMs for completing assignments. When doing so, students effectively bypass the struggle involved in the solving programming problems. Prather et al. [25] found that unconstrained use of LLMs

may be counter-productive to lower-performing students, creating an illusion of competence. We believe that educational tools to foster and assess deep understanding of programs may play a role in counteracting misleading perceptions of self-efficacy in learners.

Computing courses usually employ a mixture of theoretical and practical elements, namely practical assignments accompanied by written examinations [3]. When these include Multiple-Choice Questions (MCQs), concerns include the possibility of plagiarism or academic dishonesty [3], and the need for automated assessment [24], even more so in large-scale courses.

Questions about Learners' Code (QLCs) [14] were proposed as a form of assessing program comprehension skills. In this approach, questions (and possibly multiple-choice options for it) are formulated against a learner's own code structure and behaviour. Several contributions have been made in automating this process for different languages used in introductory programming courses (e.g., Java [28], Python [15], JavaScript [12]).

We explore whether QLC tests are a valid addition to a programming course's assessment process. We hypothesise that these tests can work alongside code-writing (i.e., writing functions to perform specific tasks) tests to facilitate the summative assessment of students' program comprehension skills. Further, since QLCs are personalised by each student's code, the possibility of cheating (e.g., copying from peers) is reduced because it is unlikely that the exact same QLC will be generated for different students. While prior work has explored QLCs in a formative context (e.g., [12, 14, 15]), we aim to assess whether QLC tests can serve as an augmentation to the summative assessment practices of CS1 courses. To this effect, we investigate the relation between students' performance in a code-writing pen-and-paper test, and a personalised test composed of QLCs generated from the students' own code submitted during the first 6 weeks of a CS1 course taught in Java. We address the following research questions:

RQ 1. How do QLC test scores relate to those of code-writing tests?
RQ 2. What are students' perceptions on summative assessment using QLCs?

We conducted a study comparing student performance on a written midterm test and a QLC test among 18 first-time enrollees in the Introductory Programming (CS1) course at our institution. The results revealed a statistically significant relation between the two sets of scores, suggesting that QLC-based assessment scores align with those of code-writing assessment. Since both types of assessment measure different, yet related, skills (code-writing vs. code comprehension ability), employing both could offer a more thorough assessment practice in introductory programming. The strong relation between the scores of both assessment types suggests comparable discriminatory power. A post-test survey indicated that students held a positive perception of the QLC test, recognising its usefulness as an assessment and self-learning tool.

2 Related Work

Traditionally, programming courses focus on teaching a programming language through a writing-focused approach, that is, students are encouraged to learn the constructs of a programming language through writing programs. However, completing assignments has been shown to not necessarily contribute to students' comprehension of underlying concepts [10, 20], and students may struggle to explain their own code even when they were successful in constructing working programs [13]. Successfully completing assignments without developing an understanding of the underlying concepts is defined by Kapur [9] as "unproductive success", for which there exists extensive research in the context of introductory

programming (e.g., [10, 11, 13, 27]). Comprehension-first teaching methods, where code reading and tracing activities precede code writing, have been demonstrated to be more effective in fostering deep understanding (e.g., Xie et al. [32]). Several studies have observed the relation between code reading and writing ability (e.g., [17, 18, 31]). Nurollahian et al. find that code comprehension ability is predictive of code-writing success [23].

Questions about Learners' Code (QLCs) consist of questions regarding constructs, patterns, or behaviour of code written by the learner [14]. QLCs typically have single or multiple-choice answers, which are easily evaluated automatically. QLCs may be generated from a combination of *static* program analysis (e.g., which constructs are employed) and *dynamic* analysis (e.g., code behaviour during execution). Dynamic QLCs usually target code tracing skills (e.g., "which values does this variable take during the program's execution?"), which have been shown by earlier studies (e.g., [4, 17, 18, 19, 31, 32]) to be deeply related to program comprehension and code writing skills.

Lehtinen et al. explored the correlation between students successfully submitting JavaScript code assignments and their understanding of the underlying concepts [13]. They found that finishing an assignment does not imply that the student understands the code they wrote or its underlying concepts, nor does it guarantee the absence of misconceptions the student might have about those concepts. The authors found that a third of students struggled to explain their own code even when they succeeded in finishing the corresponding assignments. A similar study found that students who tend to answer QLCs about their own code incorrectly have a higher tendency to drop out and more difficulty in writing programs [12]. These results might hint, as in the work of Nelson et al. [21], that students who struggle with program reading might also struggle with program writing. Students who incorrectly answered QLCs about their Python code tended to have weaker prerequisite knowledge [15].

Jask is a QLC generation system for Java code, where users submit code snippets from which static and dynamic QLCs are automatically generated [28]. In a preliminary study, their authors found that students tend to correctly answer QLCs addressing static aspects, yet may struggle when answering those related to program dynamics, revealing some difficulty in reasoning about their programs' execution. Further, students tend to react positively to the QLCs, particularly for autonomous learning purposes (e.g., for consolidating terminology).

A recent systematic review by Nurollahian et al. [22] found that most approaches for assessing student code structure focus on finding errors using automated tools, which may not capture skills such as program comprehension. Their study calls for novel assessment methods that capture such skills. QLCs, which target program comprehension, may provide an opportunity to complement existing CS1 assessment practices.

3 Background: Generating Questions about Learners' Code

QLCs are generated from *templates*, which represent questions that have not yet been generated from concrete code constructs. Conceptually, the generation of concrete QLCs may be thought of as using code constructs (e.g., types and identifiers) or behaviour (e.g., number of loop iterations, number of variable assignments) to "fill in the blanks" in a template. This may only occur if the targeted code is applicable to the corresponding type of QLC. For example, a QLC template targeting loop behaviour is only applicable to code which contains at least one loop structure. Figure 1 presents an example visualisation of a QLC template targeting variable tracing. A concrete QLC is generated by using student code to assign names and values to v, u, and $f(\text{args})$.

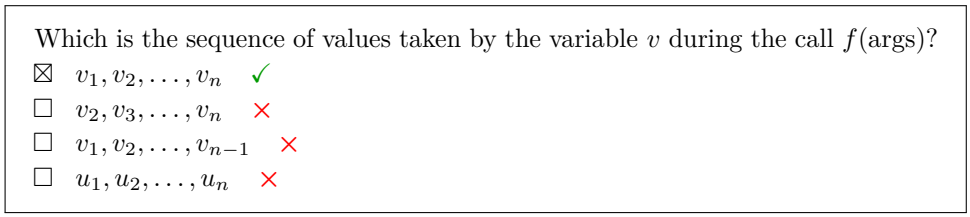


Figure 1 Example template structure for QLCs targeting variable tracing (dynamic).

Typically, QLCs contain one correct option and three distractors generated from student code. Whenever possible, these distractors are based on known introductory programming misconceptions such as boundary errors (e.g., off-by-one, as in the second and third options in Figure 1). If there is not enough data to generate distractors targeting misconceptions, options may be generated from other code constructs (e.g., values from another variable, as in the last option in Figure 1). For assignments of low complexity or those where enough data cannot be collected, broader options may be included (e.g., "none of the above"). Figure 2 presents an example QLC generated from this template, where f is materialised with `countDivisors`, while v and u are bound to its local variables `n` and `d`, respectively. The code used to generate QLCs for the purpose of our study is available as a standalone library[1].

4 Context

Our approach involves the automated generation of programming tests consisting of QLCs targeting student code previously submitted in assignments. Hence, applying it in a large-scale course requires an Automated Assessment System (AAS), or some form of collecting the students' solutions in a structured format. Figure 3 presents concepts related to AASs (grey container). Although different systems may have variations, we describe general concepts that fit most AAS realisations. A *course* holds several code *assignments*. Each assignment has several *test cases*, which associate *arguments* (inputs) to *expected* results, and define one (or more) reference *solutions* that meet its requirements. *Students* are enrolled in the course and perform *submissions* to each *assignment*. Each submission holds the student solution, which is marked with a *score*.

The remaining elements of Figure 3 relate to our approach. A *QLC test specification* is defined against the course exercises, holding several *items* that address a QLC of a certain type. Each item is associated to one or more assignments, meaning that an individual QLC will target the code that a student submitted to those assignments. A *QLC test* for a specific *student* is an instance of the test specification, comprising *questions* derived from their code. Figure 2 consists of an example of a QLC generated from a student's code.

Since different students have written different solutions to the assignments, the generated QLCs addressing that assignment will be slightly different with respect to: (a) formulation, because they refer to particular identifiers of the student solution; and (b) options, because the answers depend on the code structure and/or behaviour. Additionally, course instructors may define that a question will target a random assignment of a predefined set (e.g., with similar characteristics). This allows for more variability in the set of individual test QLCs.

[1] `https://github.com/ambco-iscte/jask`

Which is the sequence of values taken by the variable **n** during the function call **countDivisors(7)**?

```
static int countDivisors(int a) {
    int n = 1;
    int d = 0;
    while (n <= a) {
        if (a % n == 0)
            d = d + 1;
        n = n + 1;
    }
    return d;
}
```

☐ 2, 3, 4, 5, 6, 7, 8 ✗
☐ 0, 1, 2 ✗
☒ 1, 2, 3, 4, 5, 6, 7, 8 ✓
☐ 1, 2, 3, 4, 5, 6, 7 ✗

Figure 2 A concrete QLC that materialises the template of Figure 1. This example was taken from a participant's test generated for the study (translated to English).

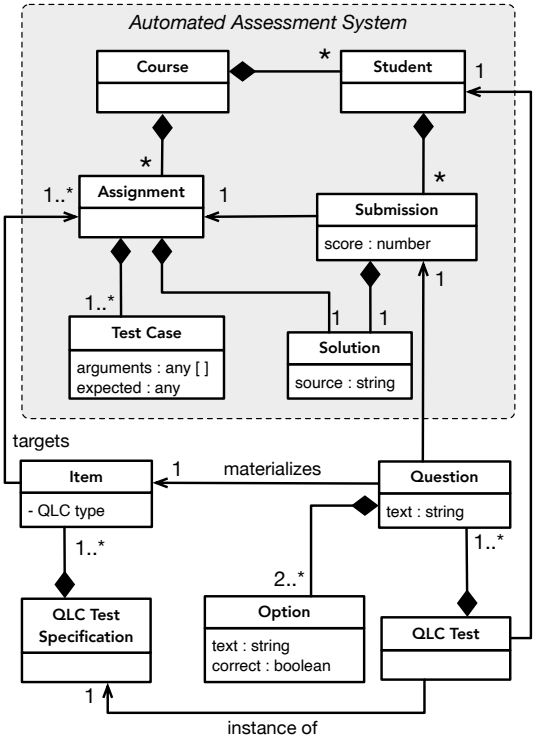


Figure 3 QLC integration within an Automated Assessment System. A test specification consists of a mapping between QLC templates and assignments. A concrete QLC test materialises a test specification using constructs from each student's code.

We introduce further variability by randomly selecting inputs from the test cases. The available inputs from tests cases, defined with meaningful values and addressing edge cases, are crucial for deriving proper dynamic QLCs that make sense for each assignment. Having a purely randomised form of generating program arguments is not adequate to address a set of assignments, as it is impossible to simultaneously guarantee that each assignment's constraints (e.g., input array must be ordered) are verified.

5 Methods

We conducted a study with first-time CS1 students at our institution. The course is taught using Java, and students are required to take a midterm test after the first 6 weeks of the course. The midterm test targets the following concepts: functions; expressions; variables; control structures; arrays; references; and procedures (side-effects). The midterm test is hand-written (pen-and-paper) and consists of questions where students write relatively small Java functions (approximately 5–15 lines of code), either by solving a problem from scratch for a set of requirements, or writing a function conforming to a given signature.

Students must complete coding assignments throughout the semester. These assignments target the same concepts as the midterm, and similarly consist in writing functions to fulfill given requirements. Code solutions to these assignments are similar in length (mean of 12 lines of code, $SD = 11$) to solutions to questions in the midterm. The QLCs generated for the study targeted these assignments.

5.1 Participants

Eighteen (18) students took part in the study. Regarding gender (self-reported), 16 participants (89%) identified as male, and the remaining two (11%) as female. Their ages ranged between 18 and 50. All the participants were enrolled for the first time in the course, but their major and previous experience differed. Most participants (14) were majoring in Computer Science and Engineering. Half of these were enrolled in the evening study program, whose student profile consists of people who have a full-time job during the day. The remaining participants were majoring in Information Systems.

The data collected for our study consisted in participants' code submissions, course midterm score, and answers to the QLC tests. Student IDs were used internally to match the different data to enable analysis, but these IDs were not stored in the resulting dataset. The data is not published nor does the paper make any mentions of identifiable student data. As per our institution's guidelines, all participants were informed of this through an informed consent form which was presented before the study. Additionally, the participants were informed that they could opt out of the study at any moment without repercussions. All 18 participants agreed to take part in the study and gave their informed consent through the signed form.

5.2 Study Sessions

The study was administered within a time frame of one week, and occurred two weeks after the participants had taken their midterm test. The one-week time frame resulted from an inability to schedule a single session which all participants could attend simultaneously.

We designed a sequence of QLC items that covered the course's assignments, and generated a personalised test for each of the participants from this specification and their respective submissions. When a participant had no submission to an assignment referenced in the test specification, we used the reference solution held in the AAS to generate the QLC.

Table 1 QLC types employed in the test. QLC types tagged with a checkmark symbol (✓) target static program aspects. Wrong measurements consider both incorrect and blank answers.

QLC Type	Template	Total	Wrong
`HowManyVariableAssigns`	How many times is the variable v assigned when calling f(args)?	18 (1/test)	11 (62%)
`HowManyFunctionCalls`	How many calls to f_2 are performed when calling f_1(args)?	18 (1/test)	7 (39%)
`WhichVariableValues`	Which are the values taken by the variable v when calling f(args)?	36 (2/test)	13 (36%)
`HowManyArrayReads`	How many array position reads are performed when calling f(args)?	18 (1/test)	5 (28%)
`HowManyLoopIterations`	When calling f(args), how many iterations does the loop perform?	54 (3/test)	14 (26%)
`HowManyArrayWrites`	How many array position writes are performed when calling f(args)?	36 (2/test)	7 (19%)
`WhichParameters` ✓	Which are the parameter names of the function f?	18 (1/test)	2 (11%)
`WhichReturnType` ✓	What is the return type of the function f?	18 (1/test)	1 (6%)
`WhatIsResult`	What is the value returned by the function call f(args)?	18 (1/test)	1 (6%)
`HowManyParams` ✓	How many parameters does the f function take?	18 (1/test)	0
`WhichVariableHoldsReturn` ✓	Which variable holds the result of the function f?	18 (1/test)	0
		270	**61 (23%)**

The participants were brought to a room with their personal laptops, which they used to conduct the test. The session started by having all participants read and sign the informed consent form. The participants were then informed about how the test would be conducted. Participants could ask clarifying questions whenever they considered that the formulation of the questions was not sufficiently clear. However, they could not pose questions regarding concepts that were addressed in the test (e.g., how loops work).

The tests consisted of 15 QLCs, each with one correct option out of four generated from the participants' code submissions. The type and number of the QLCs included in the test are presented in Table 1. All QLCs contributed 1 point towards the final score. Incorrect answers subtracted 0.25 points, and blank answers were scored with 0 points.

Participants were given 30 minutes to solve the test, whose questions were written in the teaching language of the course, employing the terminology adopted in its materials. As in the proctored midterm, the QLC test was closed-book and LLM usage was not allowed. Figure 2 presents a translated example of a QLC used in a participant's test. When each participant finished the test, or when the 30 minutes had passed, they were instructed to submit their responses, and were then shown their percentage score and the solutions.

To debrief, participants were given feedback regarding any questions they might have had regarding the questions present in their test. Finally, the participants were sent a survey in which they could express their perceptions regarding the QLC test.

5.3 Data Analysis

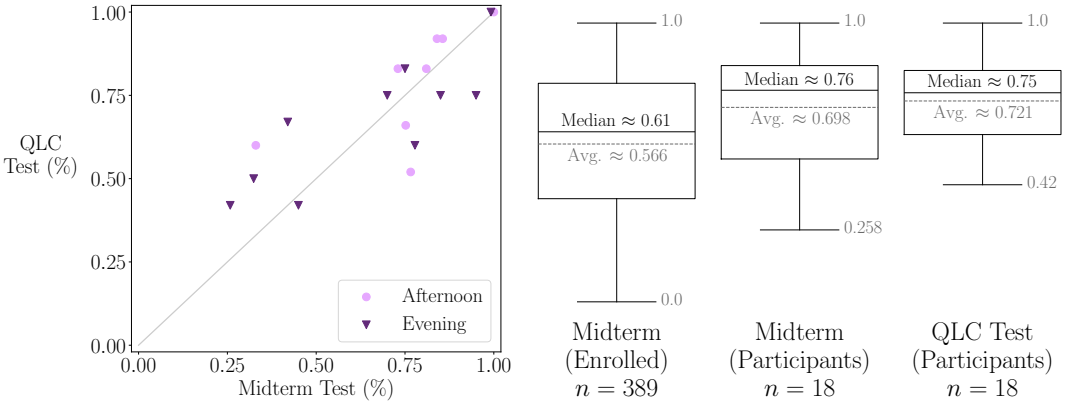
We employed linear regression to analyse the association between the QLC test and midterm scores, controlling for average code submission score and length (lines of code), and for whether the student is enrolled in the afternoon or evening study program (binary variable). The post-test survey answers were labeled through an inductive coding by one of the authors.

6 Results

Table 1 summarises the QLCs covered in the test. Approximately 23% of participants' answers were incorrect (58) or blank (3). Approximately 3% of QLCs used reference solutions due to students not having any correct submissions for the corresponding assignment.

6.1 QLC Test Results (RQ 1)

Figure 4a presents the scores (%) obtained by each participant in the QLC test in relation to their corresponding scores in the midterm test. The scores have a strong and significant correlation (Pearson $r \approx 0.784$, $p < 0.001$). The set of participants' scores exhibit nearly identical means and medians in the midterm and QLC tests, as depicted in Figure 4b. In particular, the difference in means is not significant (t-test, $p = 0.747$). However, there is a moderate and significant difference between the means of all course enrollees' midterm scores and those of the participants (t-test, $p = 0.036$, Cohen's $d = 0.51$).



(a) Relation of midterm and QLC test scores. **(b)** Score distributions per student group.

Figure 4 Relation between participants' midterm and QLC test scores.

Regression analysis shows midterm test scores along with average submission scores and lengths account for 74% of the variance in QLC test scores ($R^2 = 0.74$, adj. $R^2 = 0.66$). The intercept is not significant ($\beta = -0.460$, $p = 0.44$), nor are the coefficients for average submission score ($\beta = 1.08$, $p = 0.076$) or lines of code ($\beta = -0.232$, $p = 0.463$). The midterm test score displayed a strong and significant relation with the QLC test score ($\beta = 0.601$, $p < 0.001$). The coefficient for the student enrollment type (afternoon vs. evening program) was not significant ($\beta = -0.043$, $p = 0.445$). The coefficients represent percentages. Hence, for instance, the midterm score coefficient signals that students who score 10 percent points higher on their midterm would score 6 percent points higher on the QLC test.

QLC test scores along with submission scores and lengths offer comparable explanatory power of midterm scores ($R^2 = 0.7$, adj. $R^2 = 0.61$). The coefficient for QLC test score was positive and significant ($\beta = 1.123$, $p < 0.001$). The coefficients represent percentages scaled

to $[0, 1]$, and so the interpretation of this coefficient is that students scoring 10 percent points higher on the QLC test would score 11 percent points higher in the midterm. The remaining coefficients were not significant ($p > 0.1$).

6.2 Post-Test Survey (RQ 2)

The survey sent to each participant after the study consisted in the following questions:

1. Were the questions' statements understandable? (1: Weak, 5: Very Good);
2. Could you recognise the code in each question as your own? (1: Never, 5: Always);
3. How would you rate the duration of the test? 1: Too Short, 5: Too Long);
4. Did taking the test cause any feelings of stress, anxiety, etc.? (1: None, 5: Many);
5. Was taking the test beneficial for your learning process? How so? (Open-Ended);
6. Were there negative aspects? (Open-Ended);
7. What could have gone better? (Open-Ended);
8. Is there anything else you'd like to say? (Open-Ended).

The answers to questions Q1–Q4 are summarised in Figure 5. All participants answered Q1 positively, indicating QLCs were easily understood. For Q2, most participants were able to recognise the code in each QLCs as their own. Some participants not recognising their code may simply indicate difficulty in recalling what they submitted for earlier assignments. The answers to Q3 indicate that the 30-minute duration for 15 multiple-choice QLCs was largely adequate. The answers to Q4 indicate that taking the QLC test did not elicit negative emotions in general, with only two participants answering neutrally or negatively.

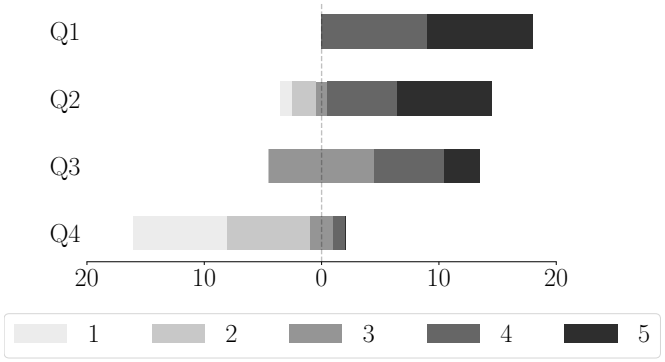


Figure 5 Post-test survey answers (Q1–Q4).

Table 2 Benefits of the QLC test perceived by the participants.

The test helped to...	# of Participants
Reveal Weak Points	10 (55.6%)
Consolidate Knowledge	9 (50%)
Increase Motivation	3 (16.7%)

The positive aspects mentioned by the participants (Q5) are summarised in Table 2. Students particularly highlighted the benefit of QLC tests for consolidating knowledge and revealing weak points in their knowledge. One participant mentioned the benefit of each QLC focusing on a particular concept, and immediate feedback as a motivating factor.

Regarding Q6 and Q7, only two participants mentioned negative aspects. One participant stated that the test elicited feelings of demotivation, as their performance challenged a misleading perception of competence. Another participant mentioned they would have benefited from having more time to take the test. In Q8, one participant suggested implementing QLC tests throughout the semester, to help consolidate concepts as they are introduced.

7 Discussion

The relatively higher failure rates on QLCs targeting program tracing confirms the results of previous studies [12, 28]. Our results additionally agree with previous observations that exercise completion does not necessarily imply comprehension [13].

Ensuring that tests are generated with enough variability is relevant for large-scale courses, as it may lower the chance of plagiarism. As with all multiple-choice questions, QLCs are prone to random guessing. We believe negative marking should be an adequate deterrent.

One of the main difficulties in automating QLCs is obtaining meaningful distractors that make sense for each question. Useful distractors are those that relate to misconceptions [26, 5], as answering QLCs may help to reveal them. Purely random generation is not adequate, as different question types or even different pieces of code for the same question might not make sense for the same types of distractors.

7.1 QLC and Midterm Test Performance Is Similar (RQ 1)

Program writing and comprehension skills are deeply correlated (see, e.g., [4, 17, 18, 19, 23, 31, 32]), and it is thus unsurprising that students who do well on code-writing assessment would do well on program comprehension questions. Nevertheless, we believe QLC tests present an opportunity to extend the summative assessment practices of CS1 courses beyond traditional approaches. The benefits would be twofold: (1) students are evaluated on their understanding of their own code, which may foster higher engagement in assignments; and (2) instructors automate the generation (and possibly evaluation) of QLC tests in large-scale courses, enabling an augmentation of existing assessment practices at scale with relatively low effort. Further, as the strong correlation of the QLC test and the midterm test scores suggests similar discriminatory power of student ability, our results suggest that QLCs may be used to augment existing summative practices to include assessment of program comprehension ability with discriminatory power similar to that of code-writing exams.

The strong relation between the scores of both assessment types is further strengthened by each being the only significant predictor in the other's regression model. The non-significance of the coefficients for submission scores and length (measured in lines of code) may indicate that the complexity and size of the submitted code do not retain their explanatory power of either assessment type's scores after controlling for the other's. This may suggest that unfairness in QLCs from students producing code of differing complexity is not a significant concern, but there are no guarantees as to how prominent this would be in a larger sample. Since the coefficient of enrollment type (afternoon vs. evening) was non-significant in both models, there should be no significant differences between students in both programs.

7.2 Students Have a Positive Perception of QLC-Based Tests (RQ 2)

Over half of the participants highlighted their perceived usefulness of the QLC test as a means to reveal their weak points, while half mentioned an opportunity to consolidate knowledge, which agrees with the observations of a prior study [28]. Our results suggest that students perceive QLCs as useful in formative and summative contexts, and could thus be used to improve student learning and assessment throughout introductory programming courses.

One study participant mentioned feeling demotivated after the QLC tests, as their performance challenged their previous (misleading) perception of self-efficacy (*"I realised I have many weak points"*). While the participant may have felt demotivated in the moment, breaking this illusion of competence provides an opportunity to improve by guiding their learning process. Misconceptions remaining "hidden" would have been harmful, both for the participant's performance in the course and the quality of their learning. Practicing with QLC tests before the course's final summative assessment may be beneficial in this regard.

7.3 Threats to Validity

The main limitation of our study is the small sample size. There is no guarantee that our results are generalisable to other contexts or institutions, or even to the overall CS1 student population in our institution. Further, there is a risk of selection bias, since students with higher self-efficacy may be more likely to participate in such studies. Our dataset showed a statistically significant and moderate difference between the average midterm scores of all course enrollees and study participants, suggesting the sample employed in our study is not representative of the broader student cohort.

Ideally, all participants would take the test simultaneously and under the same conditions. Our study was conducted over the course of one week. Thus, we cannot guarantee that participants who took the test later did not have an advantage from having more time to study the course materials. However, there is no apparent reason for this limitation to have significantly affected the results. Furthermore, given that the QLC tests were not equal (due to the randomness in the generation), it is unlikely that the earlier participants could have "leaked" any useful information regarding the questions to the following participants. There was no incentive to cheat in the study, as there was no reward related to performance.

Cheating can happen in formative contexts, and there is no guarantee that QLC success is indicative of high program comprehension – students who cheat on their assignments (e.g., by using external tools such as generative AI) may also do so on the corresponding QLCs. Usage of QLCs in summative assessment could minimise this through proctoring, for instance by having the QLC assessment coincide with the proctored written exam.

While we believe our approach should be scalable to large-scale courses, we cannot infer this from the small sample employed in our study. Summative assessment using QLCs may require extra resources, such as longer exam time and lecturers to proctor it. Formative usage of QLCs could be automated through integration in an assessment system.

While we intended to generate QLC distractors based on common student misconceptions, we did not conduct an analysis into the questions' and options' pedagogical quality. None of the study participants raised concerns regarding question interpretability, but future work could address this limitation by conducting an analysis of QLCs' quality as assessment items.

8 Conclusions and Future Work

Regression analysis of the participants' performance in QLC tests targeting program comprehension was strongly correlated to their performance in a pen-and-paper code-writing test, hinting at a strong relationship between program writing ability and program comprehension skills and comparable discriminatory power between the two forms of assessment. The results of our study suggest that QLC tests may serve as adequate augmentations of CS1 summative assessment practices, providing an opportunity to assess program comprehension.

The scope of the QLCs used in the study was limited, covering only 11 types of QLCs. We envision QLC tests to include, for example, questions targeting program repair and debugging. Further, we aim to explore the feasibility of open-ended QLCs as opposed to multiple-choice. The main difficulty in this regard would be the analysis of the answers, which may require a manual or AI-based approach. Further, students could benefit from automated feedback on their answers, eliciting potential misconceptions.

Currently, QLCs are graded dichotomously – full marks for correct answers, or none/negative for incorrect answers. In the future, we may implement partial credit in QLCs, for example by requiring students to provide open-ended justifications for their answers [2, 8]. Care must be taken to ensure this does not hinder automated grading in large-scale courses.

The test was well received by the participants, who highlighted its usefulness in consolidating knowledge and revealing weak points. Automation of QLC tests through integration in an assessment system could allow usage in formative contexts. We believe employing QLCs within both formative and summative assessment practices in CS1 courses can be an adequate augmentation towards fostering students' program comprehension skills.

Beyond summative assessment, QLC practice tests may be useful for providing formative feedback [30], as not scoring well on QLCs would point out weaknesses that would likely get revealed on summative feedback. Further, integrating QLCs into an automated assessment system (e.g., as outlined in Section 4) could enable their application after assignment completion, or periodically (e.g., weekly test), providing a formative activity where students check the comprehension of their own code. The activity is lightweight, as it should not require students to dedicate significant additional time. From our study, we conclude that 1–2 minutes may suffice to introduce a QLC targeting a specific assignment.

References

1 Afonso B. Caniço and André L. Santos. ambco-iscte/jask. Software, version 0.7.3. (visited on 2026-07-08). URL: `https://github.com/ambco-iscte/jask`, `doi:10.4230/artifacts.27040`.

2 Pablo Frank Bolton, Liberty Rose Lehr, Rahul Simha, and Michelle Lawson. The justification effect on two-tier multiple-choice exams. In *2024 ASEE Annual Conference & Exposition*, Portland, Oregon, June 2024. ASEE Conferences. `doi:10.18260/1-2--48121`.

3 W.J. Buchanan and L. Saliou. Enhanced methods of coursework provision in computer networks. In *ITRE 2004. 2nd International Conference Information Technology: Research and Education*, pages 111–115, London, UK, 2004. IEEE. `doi:10.1109/ITRE.2004.1393657`.

4 Teresa Busjahn and Carsten Schulte. The use of code reading in teaching programming. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13, pages 3–11, New York, NY, USA, November 2013. Association for Computing Machinery. `doi:10.1145/2526968.2526969`.

5 Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L. Santos, and Matthias Hauswirth. A Curated Inventory of Programming Language Misconceptions. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE '21, pages 380–386, New York, NY, USA, June 2021. Association for Computing Machinery. `doi:10.1145/3430665.3456343`.

6 James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference*, ACE '22, pages 10–19, New York, NY, USA, February 2022. Association for Computing Machinery. `doi:10.1145/3511861.3511863`.

7 James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proceedings of the 25th Australasian Computing Education Conference*, ACE '23, pages 97–104, New York, NY, USA, January 2023. Association for Computing Machinery. `doi:10.1145/3576123.3576134`.

8 Genady Ya. Grabarnik and Serge Yaskolko. Automated partial credit for stem classes. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, Covington, KY, USA, 2019. IEEE. doi:10.1109/FIE43999.2019.9028473.

9 Manu Kapur. Examining productive failure, productive success, unproductive failure, and unproductive success in learning. *Educational Psychologist*, 51(2):289–299, 2016. doi:10.1080/00461520.2016.1155457.

10 Cazembe Kennedy and Eileen T. Kraemer. Qualitative observations of student reasoning: Coding in the wild. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 224–230, New York, NY, USA, July 2019. Association for Computing Machinery. doi:10.1145/3304221.3319751.

11 Päivi Kinnunen and Beth Simon. My program is ok – am i? computing freshmen's experiences of doing programming assignments. *Computer Science Education*, 22(1):1–28, March 2012. doi:10.1080/08993408.2012.655091.

12 Teemu Lehtinen, Lassi Haaranen, and Juho Leinonen. Automated questionnaires about students' javascript programs: Towards gauging novice programming processes. In *Proceedings of the 25th Australasian Computing Education Conference*, ACE '23, pages 49–58, New York, NY, USA, January 2023. Association for Computing Machinery. doi:10.1145/3576123.3576129.

13 Teemu Lehtinen, Aleksi Lukkarinen, and Lassi Haaranen. Students struggle to explain their own program code. In Carsten Schulte, Brett A. Becker, Monica Divitini, and Erik Barendsen, editors, *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE '21, pages 206–212, New York, NY, USA, June 2021. Association for Computing Machinery. doi:10.1145/3430665.3456322.

14 Teemu Lehtinen, André L. Santos, and Juha Sorva. Let's ask students about their programs, automatically. In *Proceedings - 2021 IEEE/ACM 29th International Conference on Program Comprehension, ICPC 2021*, Proceedings/IEEE International Conference on Program Comprehension, pages 467–475, United States, May 2021. IEEE. International Conference on Program Comprehension, ICPC ; Conference date: 20-05-2021 Through 21-05-2021. doi:10.1109/ICPC52881.2021.00054.

15 Teemu Lehtinen, Otto Seppälä, and Ari Korhonen. Automated questions about learners' own code help to detect fragile prerequisite knowledge. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2023, pages 505–511, New York, NY, USA, June 2023. Association for Computing Machinery. doi:10.1145/3587102.3588787.

16 Rachel S. Lim, Sophia Krause-Levy, Ismael Villegas Molina, and Leo Porter. Student expectations of tutors in computing courses. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, pages 437–443, New York, NY, USA, March 2023. Association for Computing Machinery. doi:10.1145/3545945.3569766.

17 Raymond Lister, Colin Fidge, and Donna Teague. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *SIGCSE Bull.*, 41(3):161–165, July 2009. doi:10.1145/1595496.1562930.

18 Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the Fourth international Workshop on Computing Education Research*, ICER '08, pages 101–112, New York, NY, USA, September 2008. Association for Computing Machinery. doi:10.1145/1404520.1404531.

19 Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, pages 55–106, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3293881.3295779.

20 Sandra Madison and James Gifford. Modular Programming: Novice Misconceptions. *Journal of Research on Technology in Education*, 34(3):217–229, March 2002. doi:10.1080/15391523.2002.10782346.

21 Greg L. Nelson, Benjamin Xie, and Amy J. Ko. Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in cs1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER'17, pages 2–11, New York, NY, USA, August 2017. Association for Computing Machinery. doi:10.1145/3105726.3106178.

22 Sara Nurollahian, Noelle Brown, Anna N. Rafferty, and Eliane Wiese. A Systematic Review of Approaches for Evaluating Students' Function-Level Code Structure Knowledge with a Catalog of All Discussed Patterns. *ACM Transactions on Computing Education*, page 3787450, January 2026. doi:10.1145/3787450.

23 Sara Nurollahian, Anna N. Rafferty, Noelle Brown, and Eliane Wiese. Growth in knowledge of programming patterns: A comparison study of cs1 vs. cs2 students. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2024, pages 979–985, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3626252.3630865.

24 José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. Automated assessment in computer science education: A state-of-the-art review. *ACM Trans. Comput. Educ.*, 22(3), June 2022. doi:10.1145/3513140.

25 James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, volume 1 of *ICER '24*, pages 469–486, New York, NY, USA, August 2024. Association for Computing Machinery. doi:10.1145/3632620.3671116.

26 Yizhou Qian and James Lehman. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.*, 18(1):1:1–1:24, October 2017. doi:10.1145/3077618.

27 Jean Salac and Diana Franklin. If they build it, will they understand it? exploring the relationship between student code and performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, pages 473–479, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3341525.3387379.

28 André L. Santos, Tiago Soares, Nuno Garrido, and Teemu Lehtinen. Jask: Generation of questions about learners' code in java. In Brett A. Becker, Keith Quille, Mikko-Jussi Laakso, Erik Barendsen, and Simon, editors, *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, ITiCSE '22, pages 117–123, New York, NY, USA, July 2022. Association for Computing Machinery. doi:10.1145/3502718.3524761.

29 Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. Thrilled by your progress! large language models (gpt-4) no longer struggle to pass assessments in higher education programming courses. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ICER '23, pages 78–92, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3568813.3600142.

30 Valerie J. Shute. Focus on formative feedback. *Review of Educational Research*, 78(1):153–189, 2008. doi:10.3102/0034654307313795.

31 Anne Venables, Grace Tan, and Raymond Lister. A closer look at tracing, explaining and code writing skills in the novice programmer. In *Proceedings of the fifth international workshop on Computing education research workshop*, ICER '09, pages 117–128, New York, NY, USA, August 2009. Association for Computing Machinery. doi:10.1145/1584322.1584336.

32 Benjamin Xie, Dastyni Loksa, Greg L Nelson, Matthew J Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J Ko. A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2-3):1–49, January 2019.

33 Daniel Zingaro, Michelle Craig, Leo Porter, Brett A. Becker, Yingjun Cao, Phill Conrad, Diana Cukierman, Arto Hellas, Dastyni Loksa, and Neena Thota. Achievement goals in cs1: Replication and extension. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 687–692, New York, NY, USA, February 2018. Association for Computing Machinery. doi:10.1145/3159450.3159452.