Santos, André; Soares, Tiago; Garrido, Nuno; Lehtinen, Teemu

Jask: Generation of Questions About Learners' Code in Java

Published in:
ITiCSE '22: Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1

DOI:
10.1145/3502718.3524761

Published: 07/07/2022

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Santos, A., Soares, T., Garrido, N., & Lehtinen, T. (2022). Jask: Generation of Questions About Learners' Code in Java. In *ITiCSE '22: Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1* (Vol. 1, pp. 117-123). Article 3524761 ACM. https://doi.org/10.1145/3502718.3524761

Jask: Generation of Questions About Learners' Code in Java

André L. Santos
andre.santos@iscte-iul.pt
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL
Portugal

Tiago Soares
tiago.martinho.soares@gmail.com
Instituto Universitário de Lisboa (ISCTE-IUL)
Portugal

Nuno Garrido
nuno.garrido@iscte-iul.pt
Instituto Universitário de Lisboa (ISCTE-IUL), IT-IUL
Portugal

Teemu Lehtinen
teemu.t.lehtinen@aalto.fi
Aalto University
Finland

ABSTRACT

We present Jask, a system capable of generating questions about a learner's code written in Java. Given Java code as input, Jask provides a set of meaningful questions formulated in terms of the actual code (using its constructs and identifiers) and the corresponding correct answers. We integrated Jask in a web-based system where students submit their code (e.g., from lab exercises), answer questions about it, and obtain immediate formative feedback with the correct answers. An initial study involving 123 distinct introductory programming students providing 2274 answers revealed that questions pertaining to program dynamics tend to register low scores, possibly evidencing fragile comprehension of programming constructs. Participants were surveyed, revealing a positive view towards the usefulness of Jask, especially with respect to consolidating terminology.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Applied computing** → **Computer-assisted instruction**.

KEYWORDS

question generation, automatic assessment, program comprehension, self-explanation

1 INTRODUCTION

Learning activities addressing program comprehension gather a wide interest in the CSE community [5]. In an ideal learning process, students would fully understand the code they write, but in reality this is seldom the case [7]. A programming assignment whose output matches the expected one does not imply that students fully grasp the applied programming constructs. For instance, one may solve a problem by drawing analogies from existing code solutions, and consequently, only achieve a superficial understanding what is being coded.

In previous work we proposed the notion of Questions about Learner's Code (QLC) [11] as a learning activity to promote self-reflection and contribute to deeper comprehension of programs. These are questions obtained by static and dynamic analysis of a learner's own program (e.g., possibly submitted to an assessment system) that ask about program characteristics in terms of programming concepts (e.g., *Which is the role of variable [i] in function [f]?*, with *f* being a function authored by the learner and *i* a variable therein). Although we have discussed possible applications of the concept, no system development had been carried out.

The main contributions of this paper are the description of Jask, a QLC system for Java programs, and the results of a first evaluation of using the latter in an introductory programming course. Jask is a web-based system where students submit working code and are presented with QLCs, for which they obtain the result of their answers, getting the correct answers when they fail (see Figure 1).

Our main aim is to provide a lightweight and scalable means of *formative feedback* [17] for students to reflect on their own code and knowledge, with the intent of modifying their thinking towards improved learning. When facing an incorrect answer or a poorly understood question, a student may reinforce learning by gaining awareness of fragile mastery of concepts and terminology (e.g., programming primitives, recursion, roles of variables [15]) or misconceptions [6, 14] (refer to [1] for a catalog). To our knowledge there is no system that provides the kind of automated formative feedback we propose (refer to [8] for a survey of feedback generation for programming exercises).

We evaluated Jask by carrying out an experiment during an introductory programming course, recruiting student volunteers for using the system autonomously. We had a total of 123 participants and two rounds of questions. Whereas QLCs pertaining to concepts and terminology scored relatively high (above 80%), the QLCs scores reveal weak results in questions related to program dynamics (below 50%). A post-experiment questionnaire reveals a general positive attitude towards the usefulness of Jask, and a majority of participants indicated that they felt it contributed to strengthen their mastery of programming skills, especially with respect to terminology (indicated by 89% of respondents).

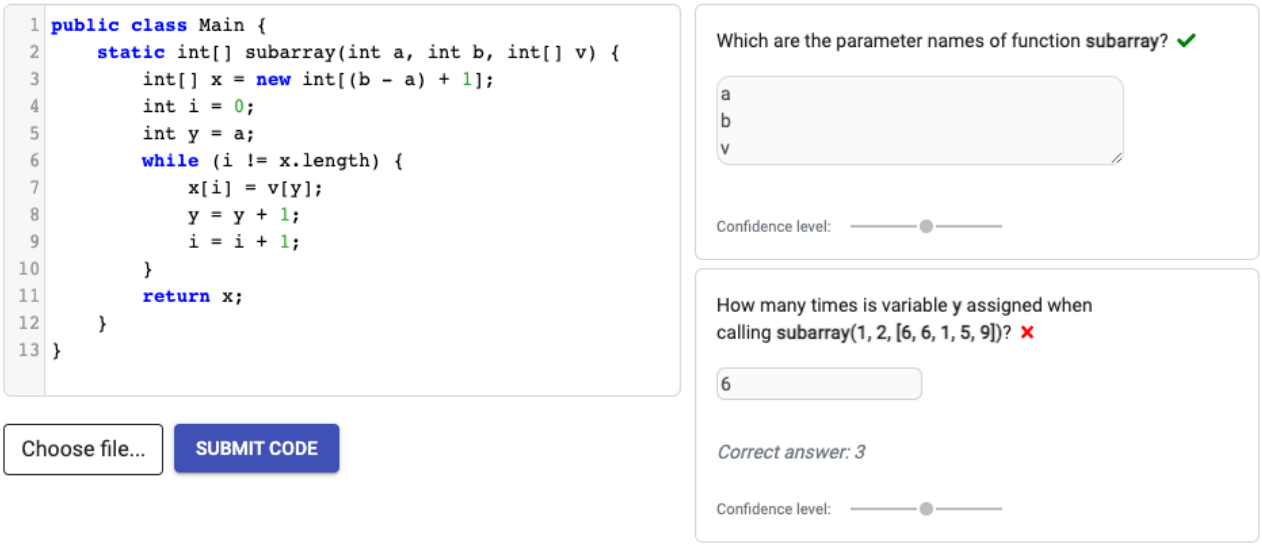

Figure 1: User interface of Jask. Input code and presentation of question results after submitting answers.

2 RELATED WORK

A qualitative study [7] closely observed the behavior of introductory programming students, revealing that they often reach a solution that outputs the correct output without fully understanding the applied programming concepts. A common strategy is based on trial-and-error until the desired outcome is achieved. This goes in line with another study that found that students may reflect negative views on self-efficacy even after a positive programming experience [9], such as a successful exercise submission. Therefore, students may go through the assignment (possibly because they have to), but they recognize the fragility of their skills for that task. Our approach aims at fostering self-comprehension of student-authored solutions by means of the QLCs. We decided to provide a feedback type of knowledge of correct response (KCR [13]), which revealed a positive effect in a programming learning context when compared to knowledge of result (KR) without informing the correct answer [3].

In previous work we have conducted a small scale experiment where QLCs were used as learning activity in an introductory course on web programming [10]. However, QLCs were prepared manually and accepted open-ended text answers (e.g., "Describe the responsibilities of your outer loop in few words."). We found that about one third of students struggle to explain their code, corroborating the findings of the previously mentioned studies [7, 9]. Further, we found that students answering QLCs correctly correlates stronger with course retention and success than students only submitting correct programs.

Henley et al. propose the notion of an "inquisitive editor" that proactively asks pop-up questions about code when a misconception is detected [4]. This approach is still in a concept development stage with no working prototype. Their approach is closely related to ours, but questions are posed earlier, during the code writing process instead of when a program is complete (i.e. suitable to be executed). Their approach seems more suitable as an advisor that warns about potential problems on the spot, rather than a post-coding activity to assess code understanding (what we aim at).

GenCODE [19] is a system to generate program tracing exercises with multiple-choice answers obtained by simulating execution to obtain distractor items. The questions address generated program snippets without any theme or meaning, comprising in isolation or combination, assignments, loops, and conditionals, solely with the purpose of asking tracing questions. In our approach, we also simulate execution for tracing-related questions, but the latter address the learners' own code instead of generated snippets. Whereas GenCODE is specialized for practicing tracing skills, we aim at a self-explanation and reflection about one's code.

3 QUESTIONS ABOUT LEARNERS' CODE

A QLC may address any aspect that relates to a piece of code, ranging from questions related to terminology, mastery of programming primitives, algorithmic strategy, relation to other pieces of code, program dynamics, etc [11]. Depending on the question's nature, some are suitable to determine the correct answer automatically, whereas others are not, particularly those with an open-ended answer (e.g., "explain the purpose of this code segment"). In this paper we focus only on types of QLCs whose answer can be obtained automatically, either by means of static code analysis, dynamic analysis, or a combination of both.

A QLC can only be meaningful if it makes sense in the context that it is posed. For instance, asking how many loop iterations are performed given a function execution should only be applicable if that same function has loop structures. As another counter-example, asking to trace the values of a variable in the context of a function that has no mutable variables makes no sense either. Therefore, some types of QLC may require that certain preconditions are met in order that the question formulations are applicable to the context.

Question type id	Question template	Function precondition
CallsOtherFunctions	Does function [f] depend on other functions?	-
HowManyFunctions	How many functions does function [f] depend on?	-
WhichFunctions	Which other functions does function [f] depends on?	There is at least one call statement to another function.
IsRecursive	Is function [f] recursive?	There is at least one call statement.
HowManyParams	How many parameters does function [f] have?	-
WhichParameters	Which are the parameter names of function [f]?	-
HowManyVariables	How many variables (not including parameters) does function [f] have?	-
WhatVariables	Which are the variables names (not including parameters) of function [f]?	-
WhichVariableHoldsReturn	Which variable will hold the return value of function [f]?	Result is given by a single variable.
WhichFixedVariables	Which are the fixed value variables of function [f]?	There are fixed value variables (local constants) being used.
WhichVariableRole	What is the role of variable [v] in function [f]? (Options: Gatherer, Stepper, Most-Wanted Holder)	There is at least one variable whose role can be determined.
HowManyLoops	How many loops does function [f] have?	There is at least one control structure.
HowDeepCallStack	What was the maximum call stack depth when calling [f(arg1, arg2, ...)]?	There is at least one call statement.
HowManyFunctionCalls	How many function executions are performed when calling [f(arg1, arg2, ...)]?	There is at least one call statement.
HowManyVariableAssignments	How many times is variable [v] assigned when calling [f(arg1, arg2, ...)]?	There is at least one variable that is assigned multiple times.
WhichVariableValues	Which is the sequence of values taken by variable [v] when calling [f(arg1, arg2, ...]?	There is at least one variable that is assigned multiple times.
WhatIsResult	What is the value returned by the function call [f(arg1, arg2, ...)]?	Return type is a value.

Table 1: Questions about Learner's Code supported by Jask. Elements in brackets denote placeholders for concrete elements obtained from submitted code. The upper section contains static QLCs, whereas the bottom section contains dynamic QLCs.

Within our scope of question generation there are two broad categories: *static* and *dynamic*. The former addresses structural aspects of code and does not require a context of program execution in order to be formulated, whereas the latter addresses behavioral aspects that require a concrete execution scenario to formulate the question and obtain the corresponding answer. While static QLCs require static analysis of the code, dynamic QLCs are more challenging, given that some form of program simulation or instrumentation is necessary. Further, dynamic QLCs require the generation of meaningful inputs to formulate the question.

Table 1 presents the QLC types that we had implemented by the time when the first evaluation of Jask was performed with student users (details in Section 5). So far, we successfully address QLCs focusing on functions (Java methods) and their inner behavior. Other types of QLC could be addressed within this scope, for instance addressing side-effects (e.g., pure functions) or recursion (e.g., base/recursive cases and tail calls).

4 JASK

4.1 User interface

Jask a web-based system in which users submit the code of a Java class and are presented with a set of QLCs addressing that same code. Upon the submission of the answers to the QLCs the user obtains the correct answer. The QLCs and both the expected (correct) and user answers are stored in a database. Additionally, we included an answer confidence (1 to 5) for each question. Figure 1 presents the key screens of the user interface of Jask, where the input code holds a function to produce a sub-array of an existing one, comprising the indexes in the range *[a, b]*. On the right hand side we present two QLCs as examples. The first is static (user result was correct), whereas the second is dynamic (user result was incorrect, and the correct answer is given).

4.2 Implementation

The process of generating questions in Jask is illustrated in Figure 2. The system has four internal components:

- *Static analyzer*. The starting point is a Java file written by the user. The code is first analyzed in order to obtain a *program model* that holds all the structural information (e.g., method

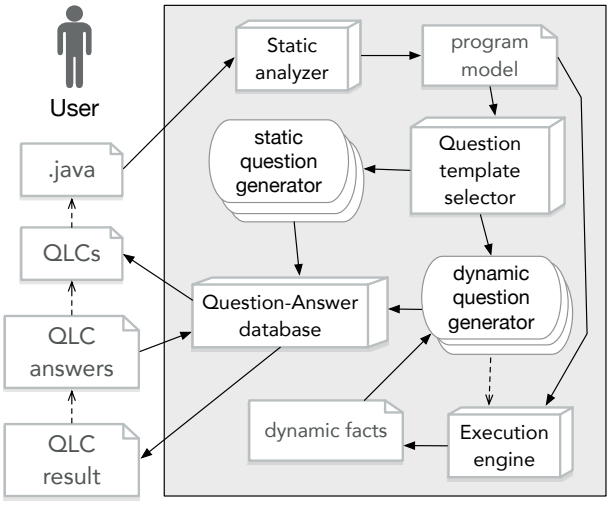

Figure 2: Process of generating questions in Jask. Dashed arrow: dependency; solid arrow: data-flow.

signatures, statements, types, etc). The program model consists of an in-memory data structure that facilitates querying and analyzing the user code, for instance to classify roles of variables automatically [16]. This is currently being achieved using a library of our own.

- *Question template selector.* This represents the complete set of currently supported QLCs. At this point, using the program model, a subset of applicable QLCs will be selected based on their preconditions. The generation of static QLCs can be performed straight away, as the program model itself holds all the necessary information to formulate the question and obtain the answer.
- *Execution engine.* The generation of dynamic QLCs requires an additional step, which consists of executing the program model through simulation to collect *dynamic facts* to obtain the answer. Distinct QLCs will require different information. However, for the QLCs we support so far, the dynamic facts consist mainly of call stack and variable state history.
- *Question-Answer database.* The generated QLCs, along with their correct answers and the source code that served as input to the generation, are stored in a database, before being delivered to the user. The user provides an answer to each QLCs, which are checked against the stored information to deliver the final result.

4.3 Limitations and future improvements

The current implementation supports the elementary Java constructs related to structured programming (assignment, sequence, selection, iteration, recursion), leaving out the possibility of using system libraries and advanced features (e.g., inheritance, lambdas). This limitation is mostly due to the fact that the *execution engine* is working with the program models. Although working with the program models facilitates the collection of dynamic facts, the need to address fully-fledged Java while maintaining semantics most likely requires executing the bytecode itself, using some form of program instrumentation. Nonetheless, Jask supports the kind of program logic that is typically approached in an introductory programming exercises.

We have used Jask experimentally in a course of our institution for research purposes, but not on a regular basis and as a mandatory learning activity. Hence, we did not address issues related to the frequency, quantity, and repetition of QLCs. In a scenario where this kind of system would be used on a weekly basis, the system behavior could have to be adapted according to student performance. For instance, if a student consistently gives correct answers a certain type of QLC, perhaps the system should stop asking it, or at least lower its frequency.

5 EXPERIMENT

As a first evaluation of Jask, we investigated how students perform in the QLC answers in the context of an Introductory Programming course. Our research questions (RQ) were the following:

(1) How do students perform in QLCs on their lab exercises?
(2) How do students perceive the activity of answering QLCs?

5.1 Context

The Introduction to Programming course is a 12-week course (CS1) taught using Java, offered by our university (University Institute of Lisbon, Portugal). In each week there is a lecture of 1.5h and a lab class of 3h where students solve code exercises. Lab classes are in groups of 20 to 25 students and guided by a teaching assistant.

Course syllabus is divided into two parts. The first 6 weeks address procedural programming, whereas the remaining of the course focuses on classes and objects. In the context of this experiment, we targeted student code written in the first part of the course. The main reason is that we did not develop any QLCs that are specific to classes, and hence, the content of the first weeks would be a better match to test the questions. Table 2 presents a summary of which contents are introduced on each week of the first part of the course, with example lab class exercises.

5.2 Method

Student volunteers were recruited by sending an email to all enrolled students in the Autumn term of 2021, offering extra points for the course final mark (1% of total for each round of participation). The extra points were given for *participation only*, no matter how students performed in terms of correct answers. Students were asked to use Jask in two rounds:

(1) By week 5, targeting week 2 exercises
(2) By week 7, targeting week 4 exercises

Participation was unsupervised, and the task of a participant consisted of accessing the website of Jask, uploading their code, and finally answering the questions (as in Figure 1). Although the questions that are visible in Figure 1 are written in English, in this experiment questions were given in Portuguese, which is the teaching language of the course, using the same terminology as in the course syllabus.

Each code submission consisted of a Java file with several static methods. The policy for selecting QLCs for each submission obeyed the following process: (1) draw a subset of QLC types (recall Table 1)

	New contents	Example exercises
1	functions, parameters, arguments, operators	check interval boundaries, check even/odd
2	if statements, loops	absolute value, power
3	function calls, recursion	find primes, factorial
4	arrays	summation, find maximum, create sub-arrays
5	procedures, references	array swaps, array sorts
6	matrices, nested loops	algebra matrix operations

Table 2: Course syllabus summary by week (first half).

that are applicable to at least one Java method of the submission; (2) for each QLC type obtained in (1), randomly select one applicable method for generating a QLC. This guaranteed that students were not getting the same QLCs for the same exercises. A small number of generated QLCs implied a very incomplete code submission that made use of a few programming constructs.

Because QLCs generated for very limited programming constructs can be much easier to answer than QLCs for programs that solve a real task, we decided to exclude those submissions that generated less than 8 QLCs from our analysis. This resulted in excluding 10 of the 112 submissions of the first round (while no submission was excluded in the second round). We offered the possibility of a third round targeting week 6 exercises, but the low number of volunteers with valid submissions at this point (8) made us decide not to investigate further.

Finally, by the end of the course, study participants were asked to fill-in a small online questionnaire, on a voluntary basis. The questionnaire items aimed at the perceptions of students regarding effort, learning, and usefulness of answering QLCs (see Table 3). Items were answered using a five-point Likert scale, where 1 represents low and 5 represents high.

5.3 Results

We collected 2274 answers to QLCs from 123 distinct students (34% of a total of 360 enrollments). Figure 3 presents the student's proportion of correct answers ($\hat{p} = k/n$) on the left and self-reported confidence on the right (RQ1). Values are given for each question type and the two rounds of asking questions. In the first round 102 students participated, whereas the second round had a lower participation of 58 students. Figure 4 presents the distribution of overall student performance in all the answered QLCs (mean score of 74.7% ± 15.5% SD).

Success rates for 4 of the last 5 QLCs, that target program dynamics, are considerably lower (below 50%) than most of the QLCs that target static aspects (above 80%). Apart from the *HowDeepCallStack*, students still had high confidence in their answers.

The final questionnaire collected 44 answers (36% of the participants), which are summarized in Table 3 (RQ2). Q3 included a set of general course concepts, which participants could select to complement their answer. These options, and the percentage of participants that selected them were: (a) terminology (89%), loops (39%), recursion (33%), variable values (28%), function definitions (17%), function calls (17%).

Figure 3: Student's success rate and self-reported confidence for each QLC type of Table 1. Round 1 is presented on darker and round 2 on lighter color.

The questionnaire results indicate that students perceive the required effort to answer QLC as moderate (Q1) with a fair level of self-reported comprehension (Q2). A significant percentage (48%) were positive or very positive regarding the learning benefits (Q3), whereas a majority of respondents (74%) considered the process of answering QLCs useful (Q4).

5.4 Discussion

The low scores on dynamic QLCs corroborates previous studies that identified that code tracing skills are problematic even for post-CS1 students [12, 18]. A couple of respondents mentioned that in few cases the concepts stated in the QLCs were not aligned with course contents. After discussing with the course coordinator, we hypothesize that the very low scores of the `HowDeepCallStack` QLC may have been negatively influenced by this. Nevertheless, in a related QLC, `HowManyFunctionCalls`, the scores were also among the lowest, but nevertheless registering high confidence levels, making us believe that misconceptions regarding the call stack execution model may be frequent among students.

The high scores on the easier QLCs were not surprising. However, there is still a significant number of students that fail on such QLCs. Also, the fact that these simpler QLCs got lower scores on the second round is apparently counter-intuitive. This may be due to

Id	Question	Mean ± SD
Q1	When compared to other course tasks, how do you classify the necessary effort to answer QLCs?	2.7 ± 0.9
Q2	When answering QLCs, how to you classify your comprehension of those questions (terms, etc)?	3.8 ± 0.9
Q3	When answering QLCs, did you feel any learning or reinforcement of programming concepts? Which sort?	3.5 ± 1.1
Q4	To what extent do you find useful answering questions about your own code?	4.0 ± 0.9

Table 3: Questionnaire results: answers on a 5-point Likert scale.

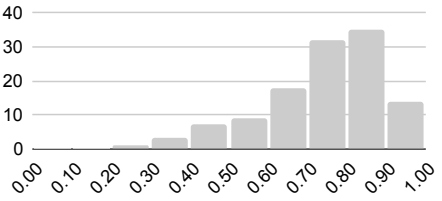

Figure 4: Distribution of individual student success rates.

the more complex exercises of week 4 (when compared to week 2), that combine multiple programming constructs. Therefore, the difficulty for the same type of QLC can depend on the source code that it targets, which is visible in comparing round 1 and 2 (clear difference at least on *WhichFixedVariables*).

5.5 Threats to validity

The recruitment of participants on a voluntary basis could have resulted in a group of subjects that is not representative of our student population in terms of programming skills. However, the distribution of individual scores (recall Figure 4) indicates a fairly balanced class subset with respect to the ability of answering QLCs.

The fact that the learning activity task was unsupervised gave us no control if the answers were actually given by the participants, and without interference from others. We addressed this threat by rewarding course points solely for participation, giving no incentive to quest for higher scores dishonestly. Further, participants might have checked other sources, such as course materials, when answering the QLCs. Although this is a positive side-effect of the learning activity, it might imply that some QLC answers do not correspond to the actual knowledge that a student had in the immediate moment when facing the questions.

6 FUTURE WORK

We plan to develop support and evaluate additional QLC types, namely addressing topics that require more complex concepts. As examples that we already managed to develop: *Does function [f] has side-effects?* (static), *The execution of [f(arg1, arg2,...)] allocates one array, what is its length?* (dynamic). In addition, we envision that QLC results could provide not only the correct answer, but also an *elaborated feedback with response contingent* [17], holding explanations targeting typical mistakes or misconceptions using the students' answers (refer to [2] for a catalog of semantic errors).

Whereas the current QLC types aim at checking if a learner understands the code, we envision another kind of QLC designed for addressing specific misconceptions. For instance, when spotting certain "anomalies" in the code structure or behavior, a QLC could drive the student to reflect on that aspect and possibly cause a student misconception to emerge when giving an incorrect answer (in a similar fashion as in [4]). In this way, a system like ours would not only promote deeper understanding, but could also spot for "silent" knowledge gaps that otherwise could remain unrevealed for longer periods.

Finally, in addition to the automatic QLC answer we would like to have an automated form of classifying incorrect answers with well-defined categories for each QLC type. For instance, the *WhichVariableValues* QLC could have the following incorrect answer types: *missing initialization*, *missing last value on loop iteration*, which were frequent incorrect answers we found when looking at the collected answers. This information could be used for instructor dashboards that would present not only the success rates for each QLC, but also which were the frequent mistakes. That would provide valuable insights to steer lecture time towards more emphasis on the weaker points of the class.

7 CONCLUSIONS

Jask stands as a proof of concept that it is feasible to automatically generate questions about a learner's code at an introductory level. In general, students tend to perform poorly when answering questions about program dynamics of their own code. Deeper understanding of program dynamics should, in principle, strengthen the programming and algorithmic skills, which are necessary for more advanced courses in the CS curricula.

We believe that a systematic adoption of Jask would at least raise student awareness of their weaker aspects, fostering the solidification of concepts and skills. The questionnaire results suggest that students have a positive view on the learning activity of answering QLCs, encouraging us to adopt the approach in regular course settings, and further carry out a larger study comprising more rounds and question types.

ACKNOWLEDGMENTS

This work was partially supported by Fundação para a Ciência e a Tecnologia, I.P. (FCT) [ISTAR Projects: UIDB/04466/2020 and UIDP/04466/2020]. We thank the anonymous reviewers for their valuable feedback.

REFERENCES

[1] Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L. Santos, and Matthias Hauswirth. 2021. A Curated Inventory of Programming Language Misconceptions. In *ITiCSE 2021: 26th ACM Conference on Innovation and Technology in Computer Science Education, Virtual Event, Germany, June 26 - July 1, 2021*, Carsten Schulte, Brett A. Becker, Monica Divitini, and Erik Barendsen (Eds.). ACM, 380–386. https://doi.org/10.1145/3430665.3456343 https://progmiscon.org.

[2] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. 2018. Common Logic Errors Made by Novice Programmers. In *Proceedings of the 20th Australasian Computing Education Conference (ACE '18)*. Association for Computing Machinery, New York, NY, USA, 83–89. https://doi.org/10.1145/3160489.3160493

[3] Qiang Hao, David H. Smith IV, Lu Ding, Amy Ko, Camille Ottaway, Jack Wilson, Kai H. Arakawa, Alistair Turcan, Timothy Poehlman, and Tyler Greer. 2021. Towards understanding the effective design of automated formative feedback for programming assignments. *Computer Science Education* 0, 0 (2021), 1–23. https://doi.org/10.1080/08993408.2020.1860408

[4] Austin Z. Henley, Julian Ball, Benjamin Klein, Aiden Rutter, and Dylan Lee. 2021. An Inquisitive Code Editor for Addressing Novice Programmers' Misconceptions of Program Behavior. In *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training, ICSE (SEET) 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 165–170. https://doi.org/10.1109/ICSE-SEET52601.2021.00026

[5] Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen Kraemer, Violetta Lonati, Claudio Mirolo, and Renske Weeda. 2019. Fostering Program Comprehension in Novice Programmers - Learning Activities and Learning Trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*. ACM, 27–52. https://doi.org/10.1145/3344429.3372501

[6] Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. 2010. Identifying Student Misconceptions of Programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*. Association for Computing Machinery, New York, NY, USA, 107–111. https://doi.org/10.1145/1734263.1734299

[7] Cazembe Kennedy and Eileen T. Kraemer. 2019. Qualitative observations of student reasoning. In *The 24th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. ACM, 224–230. https://doi.org/10.1145/3304221.3319751

[8] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Trans. Comput. Educ.* 19, 1, Article 3 (sep 2018), 43 pages. https://doi.org/10.1145/3231711

[9] Päivi Kinnunen and Beth Simon. 2012. My program is ok – am I? Computing freshmen's experiences of doing programming assignments. *Computer Science Education* 22, 1 (2012), 1–28. https://doi.org/10.1080/08993408.2012.655091

[10] Teemu Lehtinen, Aleksi Lukkarinen, and Lassi Haaranen. 2021. Students Struggle to Explain Their Own Program Code. In *ITiCSE 2021: 26th ACM Conference on Innovation and Technology in Computer Science Education, Virtual Event, Germany, June 26 - July 1, 2021*, Carsten Schulte, Brett A. Becker, Monica Divitini, and Erik Barendsen (Eds.). ACM, 206–212. https://doi.org/10.1145/3430665.3456322

[11] Teemu Lehtinen, André L. Santos, and Juha Sorva. 2021. Let's Ask Students About Their Programs, Automatically. In *29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021*. IEEE, 467–475. https://doi.org/10.1109/ICPC52881.2021.00054

[12] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. 2004. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin* 36, 4 (12 2004), 119–150. https://doi.org/10.1145/1041624.1041673

[13] Susanne Narciss and Katja Huth. 2004. *How to design informative tutoring feedback for multi-media learning*. Vol. Instructional design for multimedia learning. Waxmann, Muenster, 181–195. http://studierplatz2000.tu-dresden.de/lehrlern/pdf/artikel/feedbackdesign02.pdf

[14] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (oct 2017), 24 pages. https://doi.org/10.1145/3077618

[15] Jorma Sajaniemi. 2002. An Empirical Analysis of Roles of Variables in Novice-Level Procedural Programs. In *Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02) (HCC '02)*. IEEE Computer Society, Washington, DC, USA, 37–. http://dl.acm.org/citation.cfm?id=795687.797809

[16] André L. Santos. 2018. Enhancing Visualizations in Pedagogical Debuggers by Leveraging on Code Analysis. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research, Koli, Finland, November 22-25, 2018*, Mike Joy and Petri Ihantola (Eds.). ACM, 11:1–11:9. https://doi.org/10.1145/3279720.3279732

[17] Valerie J. Shute. 2008. Focus on Formative Feedback. *Review of Educational Research* 78, 1 (2008), 153–189. https://doi.org/10.3102/0034654307313795 arXiv:https://doi.org/10.3102/0034654307313795

[18] Simon. 2011. Assignment and Sequence: Why Some Students Can'T Recognise a Simple Swap. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research (Koli Calling '11)*. ACM, New York, NY, USA, 10–15. https://doi.org/10.1145/2094131.2094134

[19] Anderson Thomas, Troy Stopera, Pablo Frank-Bolton, and Rahul Simha. 2019. Stochastic Tree-Based Generation of Program-Tracing Practice Questions. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, 91–97. https://doi.org/10.1145/3287324.3287492