to appear in Proc. of ITiCSE 2023

Mind the Error Message: an Inverted Quiz Format to Direct Learner's Attention to Error Messages

Kazuhiro Tsunoda
k-tsunoda@prg.is.titech.ac.jp
School of Computing,
Tokyo Institute of Technology
Tokyo, Japan

Hidehiko Masuhara
masuhara@acm.org
School of Computing,
Tokyo Institute of Technology
Tokyo, Japan

Youyou Cong
cong@c.titech.ac.jp
School of Computing,
Tokyo Institute of Technology
Tokyo, Japan

ABSTRACT

Novice learners of programming tend to neglect error messages, even though the messages have a lot of useful information for solving problems. While there exists research that aims to user-friendly error messages by changing the wording and by adding visual assistance, most of them do not focus on drawing learners' attention to error messages. We propose *the enbugging quiz*, a novel quiz format that requests the learner to craft a program that produces a specified error. This paper reports our design of enbugging quizzes and reports the results of our initial experiment, where we observed positive effects on the learners' attitudes towards error messages.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**.

KEYWORDS

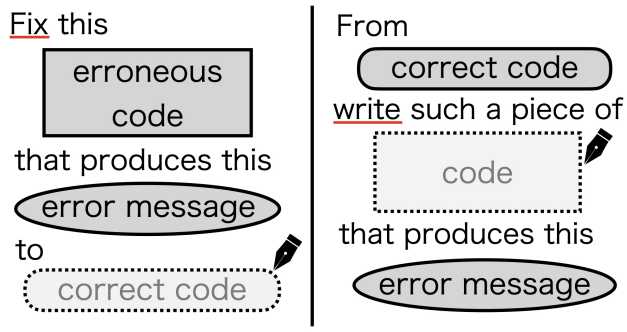
programming education, novice programmers, produce errors, language-agnostic exercise

ACM Reference Format:
Kazuhiro Tsunoda, Hidehiko Masuhara, and Youyou Cong. 2023. Mind the Error Message: an Inverted Quiz Format to Direct Learner's Attention to Error Messages. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023), July 8–12, 2023, Turku, Finland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3587102.3588823

1 INTRODUCTION

Resolving errors is one of the tasks that novice programmers feel difficult. For example, as reported by Lahtinen et al. [17], university students who took one or two programming courses considered finding bugs to be one of the three most difficult issues in programming.

Understanding error messages plays a key role in resolving errors. This is because error messages contain the information about where in the program the problem occurs (e.g., line and column numbers), which elements in the program text are relevant (e.g., variables, types and expressions), and why the error occurs (e.g., undefined names and mismatching types).



(a) debugging-style exercise (b) enbugging quiz

Figure 1: (a) Traditional debugging-style exercise and (b) proposed enbugging quiz. Given the information in the dark shaded parts, learners will fill the light shaded parts.

However, it is not easy—especially for learners—to understand, in other words, to read the above three types of information from error messages. This is because error messages have a unique syntax that is similar to but different from natural language sentences. First, error messages are often abridged [9], as in "Found: Int, Required: Double". Understanding the meaning of such messages requires translation into complete sentences, such as "There is an Int type expression in a place where a Double type expression is required." Second, error messages quote tokens from code fragments, which often confuses novices. When novices read an error message like "You typed ',' but that is not allowed", they tend to skip the comma (which is the cause of the error), because they think it is a simple punctuation mark in a natural language sentence instead of an element of their code [11]. We believe that, by reading error messages carefully, learners would be able to familiarize themselves with the syntax of error messages, just like learning the grammar of a natural language.

However, novice programmers tend to resolve errors without carefully reading error messages. This phenomenon has been reported by Umezawa et al. [28] and Chen [6]. It is also observed in the authors' preliminary experiment that tested students on how many phrases in an error message they remembered after they were shown an erroneous code fragment and the error message for a short period of time[1].

From our experience, we hypothesize the following reasons for not reading error messages:

[1]Details of the outcome can be found in the appendix, which is available at: https://prg.is.titech.ac.jp/projects/teaching-programming/enerror-generating-quiz/.

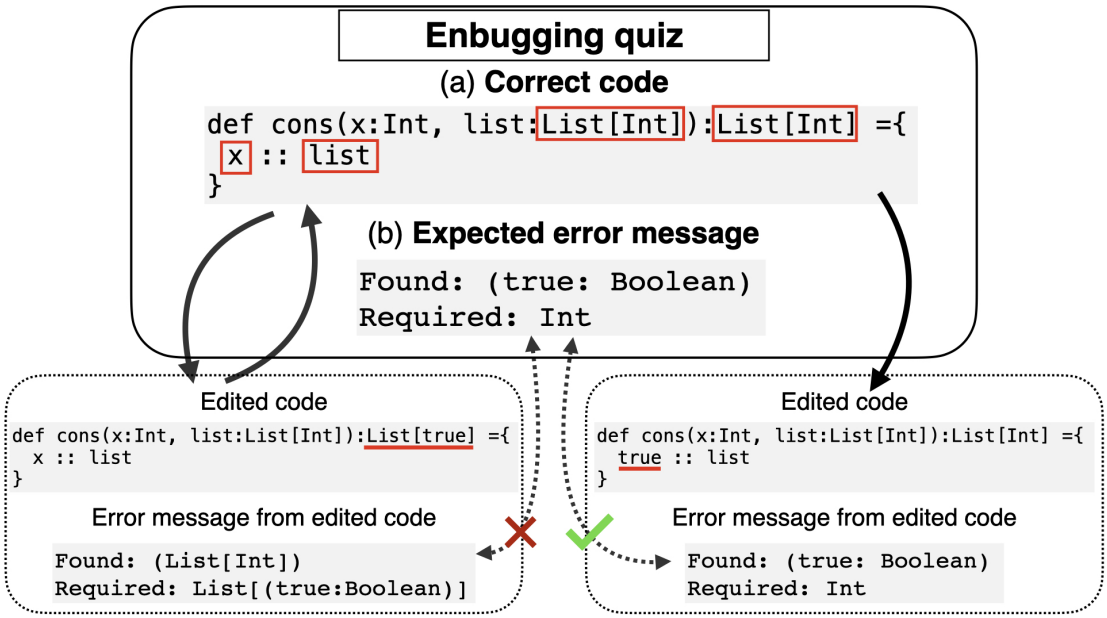


Figure 2: An example of enbugging quizzes. The learner is requested to produce the expected error message(s) by only changing the text in one of the red boxes in (a) the correct code. The first answer (bottom-left) changes the second red box, which produced an error message yet does not match the expected one. The second (bottom-right) changes the third red box, and produced the expected one.

(1) Error messages are not novice-friendly.
(2) In the early stages of learning, learners often encounter trivial errors that can be fixed without reading error messages.
(3) Not many instructors teach error messages explicitly.

Regarding the possible reason (3), we advocate that instructors should give learners an opportunity to carefully read error messages. By doing so, we expect that the learners will acquire the habit of reading error messages, which would help the learner understand error messages.

As far as the authors know, there are the following approaches to explicitly teach error messages: *teaching by example* and *debugging-style exercises*. Teaching by example is a method to show concrete pairs of erroneous code and an error message with explanations. We believe this is commonly used in novice programming courses, including the ones that the authors are teaching. Debugging-style exercises [18] are exercises in which the learners fix a piece of erroneous code (Figure 1(a)). Since the learners need to read error messages to solve problems, they are expected to learn the syntax (or format) of error messages and the terms used in them. Other than debugging-style exercises, standard programming exercises also provide opportunities to read error messages because they ask the learners to write a piece of code and the learners will inevitably make errors when writing the code.

We however see shortcomings. First, compared to learning to write code, error-resolving is less productive (in the sense that it does not create new code), and thus gives less motivation to learners. Second, since some errors can be resolved without carefully reading error messages (by trial and error, for example), debugging-style exercises might not be helpful for cultivating the habit of reading error messages. In fact, Lee et al. [19] point out that novices may skip many general (mechanical) error messages in debugging-style exercises.

We propose *enbugging quizzes* as a novel style of language-agnostic exercise for giving learners an opportunity to carefully read error messages. Enbugging quizzes ask, as shown in Figure 1(b), the learner to create a piece of code that produces a requested error message. As far as the authors are aware, there have been no prior exercises that let learners *produce* errors. We believe that this style of quiz could be useful to cultivate the habit of reading error messages since learners cannot answer this quiz without reading the error message in it.

This paper describes the design, the implementation, and a preliminary evaluation of enbugging quizzes. In the rest of the paper, we discuss related work (Section 2) and detail the format of enbugging quizzes and the design decisions behind them (Section 3). We then report our experimental use of enbugging quizzes in a programming course (Section 4) and analyze the results and feedback from students (Section 5). After discussing our plans for future extensions (Section 6), we conclude the paper (Section 7).

2 RELATED WORK

The existing studies on error messages fall into the three categories: (1) improve presentations, (2) suggest error solutions, and (3) develop exercises.

Improving presentations. As a visual improvement, Marceau et al. [22] and Funabiki et al. [10] propose and implement the highlighting of error locations on source code within IDEs (DrRacket and JPLAS). Barik [2] and Wrenn et al. [30] propose to visualize the connection between the words in an error message and the corresponding code fragments. Also, to improve the usefulness of

error messages, much research [4, 8, 26, 27, 31] implement tools that provide more detailed causes of error information within error messages and conducted experiments with students. However, some research [8, 25, 26, 31] could not confirm a positive effect of enhancing error messages like the above. Interestingly, Zhou et al. [31] noted the importance of *productive failure* from their experimental results, which could be addressed by enbugging quizzes because students intentionally make errors in the quizzes.

Suggesting error solutions. Barik et al. [3] propose to show the simplest error resolution so that novice programmers can understand the cause of the error. Hartman et al. [13] develop a technique that suggests error solutions based on similar errors and their solutions found in past editing data. Recent research by Ahmed et al. [1] confirms the improvements in the accuracy of suggestions by using neural networks. Compared to these methods, enbugging quizzes are more focused on improving students' own error-handling skills instead of teaching immediate error correction methods.

Developing exercises. The most popular exercise is debugging-style exercises, in which the learner corrects erroneous code [5, 7, 12, 18, 21]. Some research [15, 20, 24] give novices debugging-style exercises in a game style. Recent research by Kiljunen [16] suggests step-by-step debugging exercises as a form of tutorial. Those exercises provide error messages along with erroneous code, but do not require the learner to read.

3 ENBUGGING QUIZZES

In this section, we describe the design of enbugging quizzes. Throughout the paper, we use programs in Scala and error messages produced by the Scala 3 compiler (ver 3.1.1). However, as we mentioned before, enbugging quizzes can be created in other languages as well; see Section 6 for a detailed discussion.

3.1 Overview

Figure 2 gives an overview of an enbugging quiz. A quiz consists of (a) a code fragment that produces no error and has editable expressions (surrounded by red boxes), and (b) an expected error message[2] without location information (i.e., line and column numbers). A correct answer to the quiz is an edit to one of the editable boxes that make a code fragment produce the expected error message. After finishing one question (either by correctly answering or by giving up answers), the learner will see an explanation of the question. For example, below is an explanation[3] after the learner submits their answer of the quiz in Figure 2.

Correct answer Change `x` in the 2nd line to `true`.

What does the message mean? There is an expression `true` of type Boolean in a position where an `Int` expression is required.

Why is it an error? The operator `::` requires its left-hand side to have the element type of its right-hand side, i.e., `Int` of `List[Int]`. Changing `x` to `true` cause a type mismatch between `Int` and `Boolean`.

[2]In the context of enbugging quizzes, we use "error messages" to mean both error and warning messages.

[3]We prepare only one explanation even if there are multiple hypothetical solutions due to variable names, etc.

Listing 1: Example of tricky error messages

```
1 //correct code
2   def isEqual(x: Int , y: Int): Boolean = {
3   x == y
4 }
5
6 //expected error message
7 // identifier expected but ':' found.
```

3.2 Design of enbugging quizzes

A notable property of the enbugging quizzes, in comparison to debugging-style exercises, is that the learners can hardly answer questions without reading error messages. In other words, in enbugging quizzes, reading and understanding error messages plays a key role in reducing the answer space of a problem. This can be exemplified by tracing the following steps that correctly answer a question.

(1) *Interpret the expected error message.* From the expected error message (`Found: (true: Boolean)/Required: Int`), we understand that the answer code contains a `true` expression in a position where an `Int` expression is required.
(2) *Deduce an answer.* Such a case can be made by requiring an existing `true` expression to have `Int`, or by replacing an `Int` expression with `true`.
(3) *Select a box.* As there is no box with `true`, we look for boxes of type `Int`. The third box is the one because `x` is an `Int` variable.
(4) *Edit and check.* We edit the third box to `true`, and let the quiz system check. The system marks the answer correct.

In the above steps, reading and understanding the error message (1) is crucial to perform the following steps (2) and (3), which reduce the number of possible edits to a reasonable size to consider one by one. However, if we tried to answer by only superficially reading the error message (for example, just recognizing `Boolean` and `Int` but not considering the meaning of `Found` and `Required`), we would need to consider many different edits.

3.3 Design parameters

Our current enbugging quiz introduced in Section 3.1 is just one instance in the large design space. We discuss the reasons for each choice along with other possibilities.

Edit one box at a time We restrict the learner to editing only one of the boxes in order to reduce the number of possible edits to consider. An alternative is to allow editing freely, or to allow editing multiple boxes, which quickly enlarges the number of possibilities too many to handle in a short time.

Number of boxes We limit the number of boxes to 3 to 7. We think that the current limit leads to a reasonably large answer space, preventing learners from answering quizzes by randomly changing the code.

Size of code We limit the size of the code snippet to 3 to 9 lines so that each learner can quickly understand the code yet have several possibilities to consider. An alternative is to use code snippets in the real world as they are, which tend to be too large to grasp in a short time.

Location of errors We remove the location information of errors because it would often make the quiz too easy.

Kinds of error messages We have been avoiding tricky messages generated by syntax errors. Consider the quiz shown in Listing 1. The correct answer to this quiz is to change the comma in the box to a blank. However, the error message is `identifier expected but : found`. To understand this message, we need to know how the Scala compiler interprets the code, which is clearly beyond the knowledge of novice programmers.

Explanation We present an explanation after answering (or giving up) each question so that the learner can confirm their understanding of the error message. An alternative would be not to give such an explanation and expect the learners to acquire the meaning by answering many questions. We do not yet have a strong rationale for our choice.

4 EXPERIMENTS

We have conducted several feasibility experiments at the authors' institution. In this section, we describe the design and results of each experiment. Data collection and usage in these experiments have received prior consent from the students through explanation forms and consent forms.

4.1 Purpose of the experiments

The main purpose of the experiments is to assess the feasibility of the quiz format rather than its effect on learning. This is because the enbugging quiz is an unprecedented and completely unknown quiz format. The experiments investigate the following questions.

- Can students understand and work with enbugging quizzes in a real class without confusion?
- Is it possible to adjust the difficulty of quizzes so that novice students can solve them?
- Do students enjoy solving enbugging quizzes (or at least without finding it a pain)?

4.2 Preliminary experiment

As a preliminary experiment, we solicited 3 undergraduate students taking a programming course[4] and asked them to solve 6 enbugging quizzes. We observed the process of solving the quizzes in a video conference session. We also conducted a post-interview while providing explanations for quizzes that the participants failed to solve. Below are the questions we asked during the interview and the responses we received from the participants[5].

- *How did you solve the enbugging quizzes?*
  As expected, all participants went through the four steps listed in Section 3.2, namely interpret the expected error message, deduce an answer, select a box and edit and check.
- *What would you like us to add to enbugging quizzes?*
  All participants would have liked to see an explanation of the expected solutions to each enbugging quiz. In response to this feedback, we included explanations discussed in Section 3.3.

4.3 Main experiment

In the main experiment, we integrated enbugging quizzes into an undergraduate course[6] taught by the third author. The course teaches functional programming in Scala, covering basic concepts such as algebraic data types and recursion. This is the earliest class in our major that teaches programming concepts in earnest. The course had about 40 students majoring in computer science, and most of them had prior experience in Python.

This time, we incorporated enbugging quizzes into the course assignment during Week 3 to Week 6; note that there were two classes per week. The students' task in this experiment is to solve a given enbugging quiz within 5 minutes[7], using a web interface built on top of Scastie. Students were allowed to try the quiz as many times as they like before submitting their solution. When the students submitted their solution, they could see the expected solution and an explanation. We told the students that they receive points by solving enbugging quizzes, regardless of whether the solution is correct or not.

After the final exercise in Week 6, we conducted a questionnaire to collect the students' thoughts about the enbugging quiz. Below are the questions and options that students were given;

Q1 : Do you think that you will read error messages more carefully after solving enbugging quizzes?
A1 : 1. Much more frequently, 2. A little more frequently, 3. Same as before, 4. A little less frequently, 5. Much less frequently
Q2 : Do you want to use enbugging quizzes when learning a new programming language?
A2 : 1. Strongly want to, 2. Relatively want to, 4. Do not want to so much 5. Do not want to at all
Q3 : Did enbugging quizzes help you improve your error-solving skills?
A3 : 1. They helped a lot, 2. They helped a little, 4. They did not help much, 5. They did not help at all
Q4 : Do you think enbugging quizzes helped you learn programming?
A4 : 1. They helped a lot, 2. They helped a little, 3. No strong opinion, 4. They prevented learning a little, 5. They prevented learning a lot
Q5 : Do you think you are able to guess the cause of errors more frequently after solving enbugging quizzes?
A5 : 1. Quite increase, 2. A little increase, 3. Same as before, 5. Rather decrease
Q6 : Do you understand more words in error messages?
A6 : 1. Quite increase, 2. A little increase, 3. Same as before, 4. A little decrease, 5. Quite decrease
Q7 : Do you think you can understand more error messages by solving enbugging quizzes?
A7 : 1. Strongly agree, 2. Agree, 4. Disagree, 5. Strongly disagree

[4] This course was taught at the Department of Mathematical and Computing Science, Tokyo Institute of Technology in Fall 2021.
[5] The interview was held in Japanese, and the responses are translated into English by the authors when quoted in this paper. The same applies to the questionnaire in the main experiment.
[6] This course was taught at the Department of Mathematical and Computing Science, Tokyo Institute of Technology in Spring 2022.
[7] We limited the duration to 5 minutes because we concluded from our observations at the preliminary experiment that giving students more time would not affect their performance.

Table 1: Numbers of participants and correct solutions of enbugging quizzes

	day5	day6	day7	day8	day9	day10	day11	day12
Participants (people)	33	37	33	35	36	34	37	33
Correct answers (people)	24	18	26	27	14	20	20	20
Correct answer rate (%)	72.7	48.6	78.8	77.1	38.9	58.8	54.1	60.6

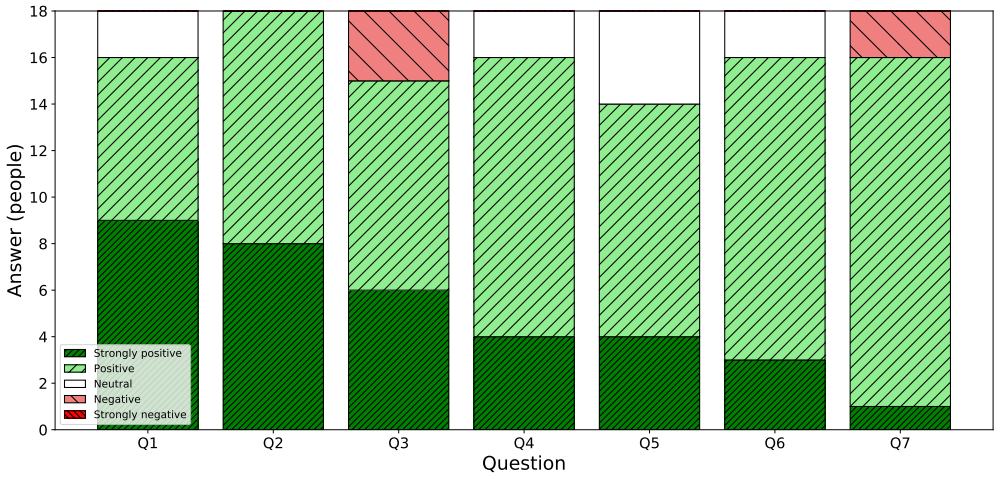


Figure 3: Results of multiple-choice questions (figure)

Table 2: Results of multiple-choice questions (table)

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
1.Strongly positive	9	8	6	4	4	3	1
2.Positive	7	10	9	12	10	13	15
3.Neutral	2	N/A	N/A	2	4	2	N/A
4.Negative	0	0	3	0	N/A	0	0
5.Strongly negative	0	0	0	0	0	0	0

We also asked the students to share their experiences and suggestions in text.

5 RESULTS AND DISCUSSION

5.1 Correct answer rate

Table 1 shows the number of participants and the number of correct answers for each quiz[8]. The correct answer rate of enbugging quizzes differs depending on the difficulty of each quiz. In particular, the correct answer rate for day 6 and day 9 is low. One possible reason for the high difficulty is that the day 6 quiz requires solid knowledge about pattern matching, which is beyond the scope of the lecture. In the case of the day 9 quiz, students probably didn't understand the functional type that they had just learned.

5.2 Results of the questionnaire

We received 18 responses to the questionnaire. The results of the selection formula are shown in Table 2 and Figure 3. The responses to the seven multiple-choice questions were mostly positive. In particular, no students thought enbugging quizzes would prevent their learning. Below we detail the responses to individual questions.

Q1: Do you think that you will read error messages more carefully after solving enbugging quizzes? Half of the responses were strongly positive, and overall there were many positive opinions. This indicates a good possibility that enbugging quizzes help cultivate the habit of reading error messages.

Q2, Q4: Effect on learning programming and new programming languages. Although the question was not directly related to error resolution, there were many positive opinions. From this result, we conjecture that enbugging quizzes may not only help students solve errors, but also increase their understanding of the programming language they are learning.

Also, in the free description about enbugging quizzes, one student wrote "I think it would be good if the code discussed in the class appeared in the problem of enbugging quizzes because it would help us understand the content of the class."

Q3: Did enbugging quizzes help you improve your error-solving skills? Three students had weakly negative opinions, but many students felt that their error-solving ability would improve. Also, more than half of the reasons for wanting to use enbugging quizzes when learning a new language were that it would help solve errors. This result supports our conjecture that enbugging quizzes would be helpful for error resolution.

Q5: Do you think you are able to guess the cause of errors more frequently after solving enbugging quizzes? More than 75% of the responses were positive. This suggests that enbugging quizzes are also effective in understanding the cause of errors.

Q6, Q7: Effect on understanding error messages. Although there are relatively few strongly positive responses, the overall percentage of positive responses is high. This result supports our conjecture that reading error messages carefully helps to understand error messages as we mentioned in Section 1.

[8] All quizzes used in the experiment are included in the appendix in footnote 1

Experiences and suggestions. Five comments indicate that solving enbugging quizzes was "fun" or "interesting" and there are no negative comments. This result suggests that students enjoyed working on enbugging quizzes.

Two comments mentioned that the explanation helped them understand problems that they could not solve. This shows the importance of the explanation.

5.3 Discussion of purpose

Based on the results of the experiment, we answer the three questions raised in Section 4.1.

- *Can students understand and work with enbugging quizzes in a real class without confusion?*
  Yes. The questionnaire results were generally positive, indicating that the students were able to work on enbugging quizzes without confusion over multiple classes.
- *Is it possible to adjust the difficulty of quizzes so that novice students can solve them?*
  Yes. The average correct answer rate for all the quizzes was about 61%, suggesting that we were able to adjust the difficulty of enbugging quizzes for novices.
- *Do students enjoy solving enbugging quizzes (or at least without finding it a pain)?*
  Yes. This can be seen from the comments received via the questionnaire.

6 FUTURE WORK

6.1 Assessment of enbugging quizzes

In the experiment we described above, we collected the participant's subjective opinions on enbugging quizzes. In the future, we would like to investigate the actual effects of enbugging quizzes on students' learning. One approach is to compare the understanding of error messages between students who have and have not solved enbugging quizzes. The other approach is to compare with debugging-style exercises, which share the key elements (code with and without errors) with enbugging quizzes. In addition to the differences discussed in Sections 1 and 2, we expect different reactions of learners when their edit does not solve the error or produce the expected error message. In debugging quizzes, it is known that novices tend to focus on eliminating the error [29], hence they may ignore the error messages produced by their edit. On the other hand, in enbugging quizzes, learners would need to read the error messages to find a correct edit.

6.2 Support for more errors

The current format of enbugging quizzes may not be effective for certain types of errors as mentioned in Section 3. On the other hand, novice programmers frequently encounter syntax errors [23] and tricky messages like the one in Listing 1. In future work, we would like to find a way to incorporate syntax errors into enbugging quizzes.

6.3 Automatic generation of quizzes

Currently, we create enbugging quizzes manually. To reduce the cost of quiz creation, we aim to automate parts of the process detailed below.

Listing 2: Indentation error in Python

```
def·equal(x,y):
··if·(x·==·y):
····→return·True
···else:
····→return·False

#error·message
#··IndentationError:·unindent·does·not·match·any·outer·
      indentation·level
```

The procedure for creating enbugging quizzes tailored to the lecture can be roughly divided into 5 parts:

- Collect code with errors that can occur in course assignment
- Classify errors
- Create erroneous code
- Specify editable boxes
- Create explanation

We would like to (semi-)automate tasks 2-4 using existing techniques. As an example, by applying the *error-repair* classification [1], we would be able to classify errors by not only error messages but also edits that fix errors. As a different example, by applying the error minimization techniques [30], we can get small erroneous codes from collected (maybe large) codes. Furthermore, by applying compiler fuzzing techniques and mutant generation [14], we can check whether there are unexpected ways of generating the expected error message for selecting editable boxes.

6.4 Support for more programming languages

To broaden the users of enbugging quizzes, it is important to create quizzes in various programming languages. While the idea of enbugging quizzes scales to any language that produces error messages, their design needs to be adjusted to each language. As an example, consider the Python program in Listing 2. The program involves an indentation error, and in this case, it is not appropriate to highlight editable boxes because generating this error requires one to insert spaces. We expect that other languages may pose different challenges.

7 CONCLUSION

In this paper, we proposed enbugging quizzes as an exercise for cultivating the habit of reading error messages. As far as we know, there are no prior exercises that let learners *produce* errors. In addition, we conducted an experiment in an undergraduate course and received positive feedback from the participants. From this result, we confirm that enbugging quizzes are feasible in programming courses despite the unique exercise format. In future work, we plan to evaluate effects of enbugging quizzes and develop a large set of enbugging quizzes by automating the creation of quizzes. We hope that enbugging quizzes would allow learners to have a better debugging experience in everyday programming.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI grant numbers 20K21790 and 23H03368. We thank the members of the Programming Research Group for their trial uses of the enbugging quizzes before we conducted the preliminary experiments.

REFERENCES

[1] Umair Z Ahmed, Renuka Sindhgatta, Nisheeth Srivastava, and Amey Karkare. 2019. Targeted example generation for compilation errors. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 327–338.

[2] Titus Barik. 2014. Improving error notification comprehension through visual overlays in IDEs. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 177–178.

[3] Titus Barik, Jim Witschey, Brittany Johnson, and Emerson Murphy-Hill. 2014. Compiler error notifications revisited: an interaction-first approach for helping developers more effectively comprehend and resolve error notifications. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 536–539.

[4] Brett A Becker. 2016. An effective approach to enhancing compiler error messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 126–131.

[5] Elizabeth Emily Carter. 2014. An intelligent debugging tutor for novice computer science students. (2014).

[6] Der-Thanq Chen. 1997. Learning 10 computer programs in a month. (1997).

[7] Ryan Chmiel and Michael C Loui. 2004. Debugging: from novice to expert. *Acm Sigcse Bulletin* 36, 1 (2004), 17–21.

[8] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing syntax error messages appears ineffectual. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*. 273–278.

[9] Paul Denny, James Prather, Brett A Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B Powell. 2021. On Designing Programming Error Messages for Novices: Readability and its Constituent Factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[10] Nobuo Funabiki, Yukiko Matsushima, Toru Nakanishi, Kan Watanabe, and Noriki Amano. 2013. A Java programming learning assistant system using test-driven development method. *IAENG International Journal of Computer Science* 40, 1 (2013), 38–46.

[11] Marleen Gilsing and Felienne Hermans. 2021. Gradual Programming in Hedy: A First User Study. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–9.

[12] Yoshinari Hachisu, Atsushi Yoshida, and Kiyoshi Agusa. 2017. A generator for Exercises of Program Error Correction and Answer Checker Programs (in japanese). *IPSJ Transactions on Computers and Education (TCE)* 3, 1 (2017), 64–78.

[13] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R Klemmer. 2010. What would other programmers do: suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1019–1028.

[14] Ryota Kamei, Masahiro Yosizuka, Isao Sasano, and Seiichi Komiya. 2021. Comprehension Check Problem Generation Method Using Perturbation to Compensate for Shortcomings of Tracing Learning—Evaluation of Effectiveness Based on Examples of Binary Operators(Syakyou gata gakusyu no kettenn wo oginau setsudou wo motiita rikaido kakunin mondai seisei syuhou—nikou ennzannsi no rei ni motoduku yuukousei no hyouka (in Japanese).). *Computer software* 38, 1 (2021), 1_111–1_139.

[15] Cagin Kazimoglu, Mary Kiernan, Liz Bacon, and Lachlan Mackinnon. 2012. A serious game for developing computational thinking and learning introductory computer programming. *Procedia-Social and Behavioral Sciences* 47 (2012), 1991–1999.

[16] Olli Kiljunen. 2021. Teaching Students to Fix Programming Errors with Tutorials Embedded in an IDE. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research* (Joensuu, Finland) *(Koli Calling '21)*. Association for Computing Machinery, New York, NY, USA, Article 32, 3 pages. https://doi.org/10.1145/3488042.3489969

[17] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A study of the difficulties of novice programmers. *Acm sigcse bulletin* 37, 3 (2005), 14–18.

[18] Greg C Lee and Jackie C Wu. 1999. Debug It: A debugging practicing system. *Computers & Education* 32, 2 (1999), 165–179.

[19] Michael J Lee and Amy J Ko. 2011. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research*. 109–116.

[20] Zhongxiu Liu, Rui Zhi, Andrew Hicks, and Tiffany Barnes. 2017. Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education* 27, 1 (2017), 1–29.

[21] Andrew Luxton-Reilly, Emma McMillan, Elizabeth Stevenson, Ewan Tempero, and Paul Denny. 2018. Ladebug: An online tool to help novice programmers improve their debugging skills. In *Proceedings of the 23rd annual acm conference on innovation and technology in computer science education*. 159–164.

[22] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. 2011. Measuring the effectiveness of error messages designed for novice programmers. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. 499–504.

[23] Davin McCall and Michael Kölling. 2019. A new look at novice programmer errors. *ACM Transactions on Computing Education (TOCE)* 19, 4 (2019), 1–30.

[24] Michael A Miljanovic and Jeremy S Bradbury. 2017. Robobug: a serious game for learning debugging techniques. In *Proceedings of the 2017 acm conference on international computing education research*. 93–100.

[25] Marie-Hélène Nienaltowski, Michela Pedroni, and Bertrand Meyer. 2008. Compiler error messages: What can help novices?. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*. 168–172.

[26] Raymond S Pettit, John Homer, and Roger Gee. 2017. Do Enhanced Compiler Error Messages Help Students? Results Inconclusive.. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 465–470.

[27] James Prather, Raymond Pettit, Kayla Holcomb McMurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. 2017. On novices' interaction with compiler error messages: A human factors approach. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. 74–82.

[28] Katsuyuki Umezawa, Makoto Nakazawa, Masayuki Goto, and Shigeichi Hirasawa. 2019. Development of debugging exercise extraction system using learning history. In *2019 IEEE Tenth International Conference on Technology for Education (T4E)*. IEEE, 244–245.

[29] Iris Vessey. 1985. Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies* 23, 5 (1985), 459–494.

[30] John Wrenn and Shriram Krishnamurthi. 2017. Error messages are classifiers: a process to design and evaluate error messages. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 134–147.

[31] Zihe Zhou, Shijuan Wang, and Yizhou Qian. 2021. Learning From Errors: Exploring the Effectiveness of Enhanced Error Messages in Learning to Program. *Frontiers in Psychology* 12 (2021). https://doi.org/10.3389/fpsyg.2021.768962