Development of Debugging Exercise Extraction System using Learning History

Katsuyuki Umezawa
Department of Information Science
Shonan Institute of Technology
Kanagawa, Japan
umezawa@info.shonan-it.ac.jp

Makoto Nakazawa
Department of Industrial Information Science
Junior College of Aizu
Fukushima, Japan
nakazawa@jc.u-aizu.ac.jp

Masayuki Goto
Department of Industrial Management and Systems Engineering
Waseda University
Tokyo, Japan
masagoto@waseda.jp

Shigeichi Hirasawa
Research Institute for Science and Engineering
Waseda University
Tokyo, Japan
hira@waseda.jp

***Abstract*—We have proposed an editing history visualization system which can confirm where and how the learner modified program. We utilized this system for actual flipped classroom and stored a large amount of learning logs. This learning log contains all the source code in the process of being modified until the program is completed. We developed a debugging exercise extraction system to automatically generate problems for debugging practice from this learning log. The debugging exercise extraction tool we developed extracted 18,680 source codes (which became practice problems) that included syntactic errors that could be used as a debugging exercise from 16 weeks of program edit history data (total number is 31,562 files). The execution time was 488 seconds. Since it can be analyzed only once every six months, we believe it is a sufficiently practical execution time.**

***Keywords*—e-Learning, Self-Study, Artificial Teacher, Language Leaning, Learning History**

I. Introduction

We have proposed an editing history visualization system that is a learning environment for programming language learning [1]. This system easily prepares the learning environment and confirms the learning situation. Also, since this system accumulates learning logs, we can see where and how the learners modified the program. Further, we proposed an effective flipped classroom based on the log information of self-study, called a "grouped flipped classroom" . In applying this grouped flipped classroom to an actual lesson, we utilized the editing history visualization system [2]. As a result, a large amount of learning logs were accumulated when about 90 students took classes for 16 weeks.

I actually did a programming lesson and found that there were several students who asked faculty members for help without reading error messages even though they were displayed. The edit history visualization system accumulates all the source code of the process that is being modified until the program is completed. We thought that by automatically extracting the source code containing errors from all the source code, we could let learners practice correcting the mistakes. We developed a tool to realize it.

A method for automatically generating fill-in-the-blank problem and error correction problem are known [3]. However, there is no method to automatically generate an error correction problem based on the raw data actually coded by the learner. Since our proposed method is based on raw data, it has the advantage that error correction problems can be automated based on other learners' real mistakes. Furthermore, it has the advantage of being applicable to any programming language.

II. Debugging Exercise Extraction System

A. Overview of Our Proposed System

The overall configuration of the debugging exercise problem extraction system developed this time is shown in Figure 1. As shown in Figure 1, the existing editing history visualization system is used until the learning history is accumulated. The debugging exercise extraction tool refers to the learning history accumulated by the editing history visualization system, compares the correct source code with the source code containing errors, and extracts the source code containing errors. The source code containing errors extracted in the previous step is referred to and distributed in a modified version of the editing history visualization system.

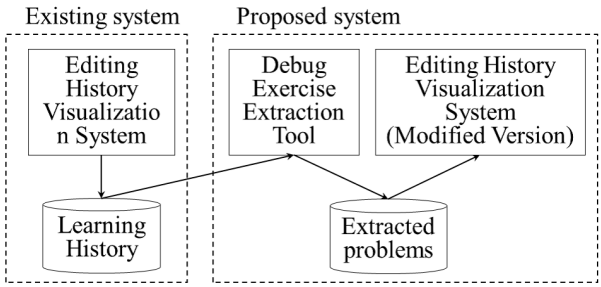


Fig. 1. Overall configuration of the debugging practice problem extraction system

B. Debugging Exercise Extraction Tool

The debugging exercise extraction tool compares the learning history accumulated by the editing history visualization system, specifically, MyClass.java existing in the complete folder (last) for each problem, and MyClass.java in the process of coding. It extract the number of mistakes and the number of misspelled characters for each mistake. The tool reconstructs the folder based on the extracted "number of mistakes" and "number of misspelled characters". There are two differences between source code: one is for correcting an error and another without error. Therefore, only source code for correcting an error is extracted.

The developed debugging exercise extraction tool is shown in Figure 2. As shown in Figure 2, we can specify how many mistakes are to be extracted, how many misspelled characters to be extracted, and which problems of lesson are to be extracted.

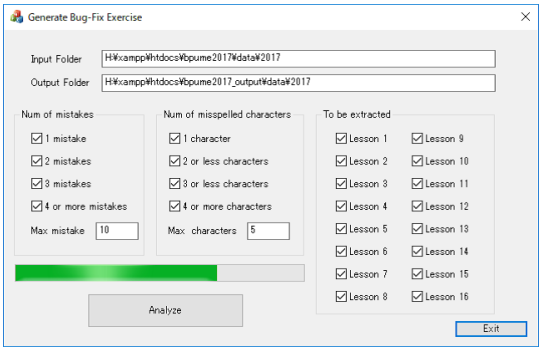


Fig. 2. Debugging exercise extraction tool

C. Extraction Algorithm

The extraction algorithm is shown in Figure 3. This proposed algorithm creates the folder structure shown in Figure 4 (b) from the folder structure shown in Figure 4 (a).

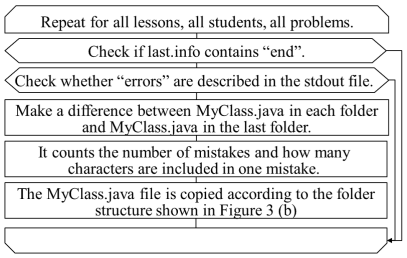
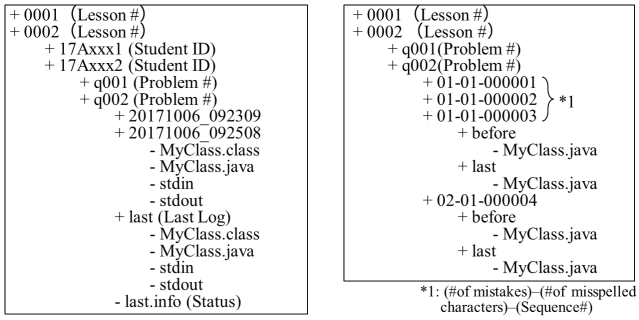


Fig. 3. Proposed Algorithm



(a) Folder structure of Editing history visualization system

(b) Folder structure of Debugging exercise extraction tool

Fig. 4. Folder structure

D. Evaluation of Execution Time for Extraction

The total number of program editing history data for this 16 weeks was 31,562 files. Among them, 18,680 files were accompanied by errors. When the debugging exercise extraction tool was executed on the computer with Intel® Core^{TM} i5-2400 3.10GHz CPU, 16GB memory, and Windows 10 Pro (64 bit) OS with the extraction option shown in the figure 2, the execution time was 488 seconds. Since it is only necessary to analyze the entire learning history of about 100 learners for half a year, it can be said that this execution time is sufficiently practical.

E. Visualization and Distribution of Extracted Data

The extracted data can be visualized with the modified version of editing history visualization system as shown. Since the editing history visualization system also has a source code distribution function, it is possible to distribute the source code with errors to students and ask questions about correcting errors.

III. Conclusion

We developed a debugging exercise extraction system to automatically generate problems for debugging practice from learning log. The proposed tool extracted 18,680 source codes (which became practice problems) that included syntactic errors that could be used as a debugging exercise from 16 weeks of program edit history data (total number is 31,562 files). The execution time was 488 seconds. Since it can be analyzed only once every six months, we believe it is a sufficiently practical execution time. In the future, we will establish an extraction method that takes logic errors and mistakable problems into consideration. In addition, we will verify that if students practice debugging using the problem extracted by our system, their dependence on teachers will be reduced.

Acknowledgment

Part of this work was supported by JSPS KAKENHI Grant Number JP19H01721, JP17K01101 and JP16K00491, and Special Account 1010000175806 of the NTT Comprehensive Agreement on Collaborative Research with Waseda University Research Institute for Science and Engineering. Research leading to this paper was partially supported by the grant as a research working group "ICT and Education" of JASMIN.

References

[1] M. Aramoto, M. Kobayashi, M. Nakazawa, M. Nakano, M. Goto and S. Hirasawa, "Learning Analytics via Visualization System of Edit Record − System configuration and implementation," [in Japanese], 78th National Convention of Information Processing Society of Japan, Vol. 4, 2016, p.p. 527–528.

[2] K. Umezawa, T. Ishida, M. Nakazawa and S. Hirasawa, "Application and Evaluation of a Grouped Flipped Classroom Method," Proceeding of the IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE2018), 2018, p.p. 39–45.

[3] H. Nagataki, R. Itoh, F. Ooshita, H. Kakugawa and T. Masuzawa, "A Fault Injection Method for Generating Error-correction Exercises in Algorithm Learning," Journal of Information Processing Society of Japan (IPSJ) 49(10), 2008, pp. 3366–3376.