Ogata, H. et al. (Eds.) (2015). Proceedings of the 23rd International Conference on Computers in Education. China: Asia-Pacific Society for Computers in Education

Construction of an Environment to Support Learning Systematic Debugging Process with Worksheets and Synchronized Observation Tool

Raiya YAMAMOTO[a*], Yasuhiro NOGUCHI[b], Satoru KOGURE[b], Koichi YAMASHITA[c], Tatsuhiro KONISHI[b] & Yukihiro ITOH[d]
[a]*Graduate School of Science and Technology, Shizuoka University, Japan*
[b]*Faculty of Informatics, Shizuoka University, Japan*
[c]*Faculty of Business Administration, Tokoha University, Japan*
[d]*Shizuoka University, Japan*
[*]dgs15009@s.inf.shizuoka.ac.jp

Abstract: In programming exercises, some novice learners cannot finish the work, even with unlimited time. For such learners, many debugging assist tools have been developed. However, those tools cannot resolve typical problems of the novice learners, i.e. (1) they cannot evaluate correctness of behaviors of their programs with information from the tools and (2) they don't know correct debugging process. In this paper, we propose a learning environment for debugging with "Worksheets" for learning a correct debugging process and "Synchronized observation tool" for visualizing behaviors of programs in a domain world. We carried out a preliminary experiment to evaluate learning effectiveness of our system.

Keywords: Programming education, Debugging, Interactive learning environment, Visualizing

1. Introduction

In programming exercises, some novice learners cannot finish their assigned work, even with unlimited time. Two general problems may cause this situation.

-Problem 1: They cannot manage information from standard debugging assist tools (for example, Yokomori et al., 2001) properly to evaluate the correctness of behaviors of their programs.

-Problem 2: They do hit-or-miss debugging because they don't know correct debugging process.

For Problem 1, many tools that assist programming learners by visualizing the behavior of their programs have been developed (Imaizumi et al., 2011, Moreno et al., 2004, Hanson et al., 1997) Such tools could be effective for some novices, but not all. They fail for the following two reasons:

-Reason1-1: Algorithm and specification of program often include concepts such as roles of data structures (for example, stack pointer), spatial feature of data structures (for example, bottom of stack, root of tree) and relative positional relations among data structures (for example, parent and child of tree). In this paper, we call a world that consists of such concepts "Domain World". However, these tools basically visualize data structures directly that the programming language supplies (for example, variable, array).

-Reason1-2: Novice learners cannot evoke correct behavior so they cannot evaluate correctness of behaviors of their programs with visualized behaviors.

In response to the above reasons, we have developed a tool that can visualize the behavior of learners' program and correct program (corresponds to reason 1-1) and can visualize both behaviors in parallel (corresponds to reason 1-2). They can compare the behavior of their program and correct program using this tool. We call this tool "Synchronized Observation tool" (Tobsync: Tool for OBservation with SYNChronized view). This tool resolves problem 1.

For Problem 2, there is a learning support tool for debugging called DESUS (Egi et al., 2009) However, DESUS assists trace learning of functions in learners' programs. DESUS does not assist learning the whole debugging process. Therefore, we propose a worksheet for learning a systematic debugging process (hereinafter referred as to LSDP-WS: Learning the Systematic Debugging Process

WorkSheet). Each step of the systematic debugging process is mapped on LSDP-WS. Novice learners can learn the systematic debugging process by working with LSDP-WS repeatedly. Tobsync is embedded in LSDP-WS. These resolve problem 2.

In this paper, we have developed a learning environment with Tobsync and LSDP-WS. Then we carried out a preliminary experiment to evaluate learning effectiveness of our system.

2. Previous Research

Tobsync should visualize behaviors of programs in Domain World. Kogure et al. have already developed a system with such feature (Kogure et al., 2012). This system analyzes a sample code prepared by a teacher. The system replicates statement execution history and visualizes the behavior of the sample code in Domain World. While watching the behavior, learners look for groups of executed statements that has a certain function in the statement execution history. Then they name the found group according to the function of the group in natural language. Through this procedure, learners understand the behavior of program. We apply Kogure's method to visualize the program to Tobsync.

3. Systematic Debugging Process and Novice Learners' Problems

We designed the systematic debugging process based on debugging-by-deduction and backtracking (Myers et al., 2012). Systematic debugging process is as follows. In step 1, programmers list up the main function and sub functions in their source code. Then, they arrange functions' calling relations to make structure model. After that, they confirm correctness of structure model they made by reading source code. In step 2, first, they select a function to be tested. In order to select the function, they adopt a selection policy like bottom up test or top down test etc. After selecting the function, they prepare input data for test and expected output corresponding to the input data. After this, they execute the selected function and compare expected output and actual output. Then, they consider hypothesis on bug(s) in the function. Finally, according to the hypothesis, they decide what to do next: whether move to another function's test or move to detailed check of the function. In step 3, they check behavior of each statement with comparing to correct behavior. When they find a statement that causes unexpected behavior, they start to check statement reversely from the statement where unexpected behavior was observed. In step 4, they fix specification or code for the point(s) found at step 3.

Table1 shows novice learners' problems for each step and solutions with our system. LSDP-WS is solution for Problem {A} and {B}, Tobsync is for [C] and [D]. Problem (E) is difficult for our current system to address directly. However, we suppose that this system can prevent learners from repeating hit-or-miss debugging because the learners can narrow set of functions and/or statements that might include the bug(s) by step 1-3. It means this system guides them to focus on inappropriate functions and/or statements.

Table1: Novice learners' problems for each step.

Step	Problems
Step1: Ordering functions' calling relations.	{A} Learners cannot recall how to arrange functions' calling relations.
Step2: Selecting function that might have bug(s)	{B} Learners cannot recall how to select function to be tested.
Step3: Selecting possible bug area in the function	[C] Learners cannot imagine the behavior of a selected function. [D] Learners cannot find difference between behaviors of their program and correct program.
Step4: Correcting specifications or source code	(E) They don't know where to fix or what to fix to correct their programs' behavior.

4. The Method of Supporting Learners

4.1 Support with LSPD-WS

Learners can learn systematic debugging process by practicing with LSPD-WS repeatedly. LSPD-WS includes sequential numbers that lead the learners to follow the systematic debugging process. The numbers are allocated to buttons and working areas.

4.1.1 LSPD-WS for learning how to arrange the calling relations of functions

Novice learners learn the following procedure how to understand structures of their programs with WS1 that is seen in Figure 1: Phase 1 is listing functions in source code, phase 2 is arranging the functions' calling relations to make structure model, and phase 3 is confirming the correctness of structure model they create. On WS1, there are buttons and areas that correspond to each phase (Figure1). Concretely, button 1 starts the phase1 ((1) in Figure1). Working area 2 suggests learners to perform the phase2 ((2) in Figure 1). Button 3 makes the system to start the phase 3 ((3) in Figure1).

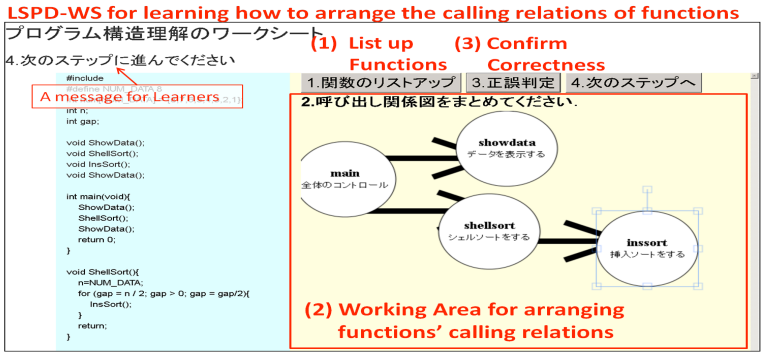

Figure 1. A user interface of WS1

4.1.2 LSPD-WS for learning how to select a function that might have bug(s)

Novice learners learn the following procedure how to select a function that might have bug(s) with WS2 that is seen in Figure 2: Phase 1 is selecting functions to be tested, phase 2 is making input data for test, phase 3 is making expected output for the input data, phase 4 is comparing the expected output to actual output, and phase 5 is considering and describing hypothesis. On WS2, there are buttons and areas that correspond to each phase, similar to WS1 (Figure2). Concretely, working area 1,2,3 suggest learners to perform the phase 1,2,3 ((1), (2), (3) in Figure2) respectively. Button 4 makes the system to start the phase 4 ((4) in Figure2). Button 5,6 starts the phase 5,6 ((5), (6) in Figure2) respectively.

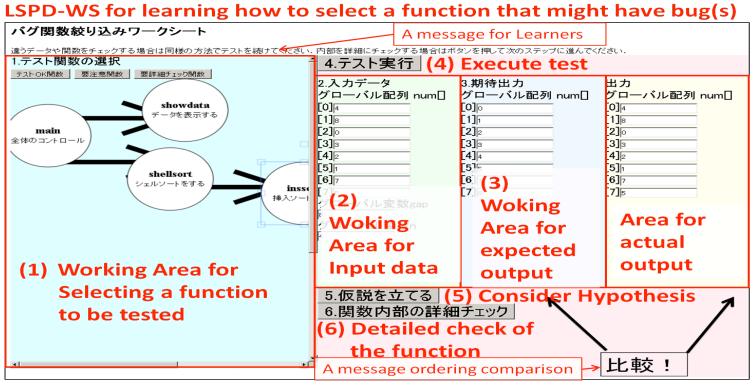

Figure 2. A user interface of WS2

4.2 Synchronized Observation in Domain World (Tobsync)

Tobsync visualizes the behavior of programs in Domain World. Area (1) is Domain World for visualizing learner's program and area (2) is for visualizing correct program. By pushing 'prev' and 'next' buttons shown in (3) and (4), learners can move visualized point in the execution history and can observe behavior of the program (Solution for Step 3 [C]). Thus, the learner can compare the behaviors between their program and correct program (Solution for Step 3 [D]).

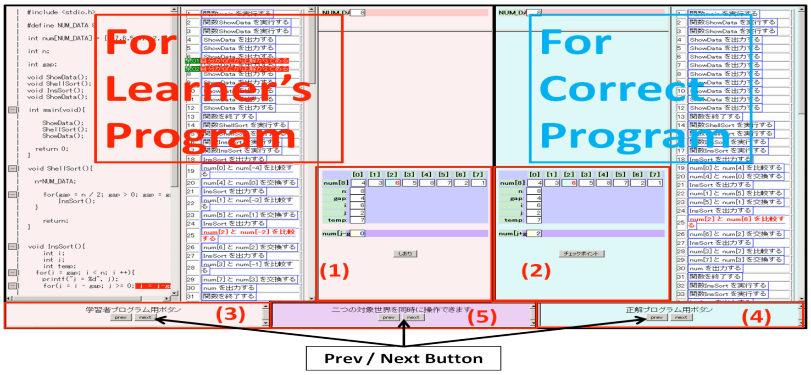

Figure 3. A user interface of Tobsync

5. Experimental Evaluation of Learning Effectiveness

5.1 Hypotheses

We have three hypotheses that confirm the learning effects of our system.
-Hypothesis 1: Learners can learn systematic debugging process
-Hypothesis 2: Observation with Domain World is effective for learning debug process
-Hypothesis 3: Synchronized observation is effective for learning debug process

5.2 Experiment overview

We gathered two types of participants. Type A learners (3 people) have learned user defined functions of programs. They are students of a department of computer science who have learned programming for a year and a half. Type B learners (7 people) have learned programming only with main function. They are students of a department of business who have learned programming for a half year. Type A learners evaluate step 1-3 and Type B learners evaluate only step 3 with our system.

To begin this experiment, the learners had a pretest to establish their level. After pretest, learners practiced three problems with our learning system. Finally, the learners had a posttest to establish their level after learning with our system. Type A learners debugged Program of Brute-force search algorithm in pretest, three shell sort programs that have different bug in practice Problem, and Program of Boyer-Moore string search algorithm in posttest. Type B learners debugged Programs that adds values in array in pretest, three different programs (Program that adds values in array, Program that calculates average of values in array, and Program that convers decimals to binaries) in practice program, and programs that reverses values in array in posttest. Learners did pretest and posttest with the editor and compiler that they usually use. In practice problems, learners practice systematic debugging process with our system to find statements that have bugs. In the pretest and posttest, the learners tried to find bugs and modify the source code to fix the bugs. In pretest, posttest, and practice problems, source codes with only one bug are used. We designed the difficulty of each source code based on the types of learners for they cannot find a bug only by reading source code. Table 2 shows the given problems by the types of learners.

For evaluation, we recorded the PC screen during the experiment and observed whether there are changes in learners' behavior between pretest and posttest. Hereinafter, we call these learners' changes "behavioral changes". In addition, learners answer the questionnaire. We also checked the results of the pretest and posttest.

5.3 Results

5.3.1 Results from videos

As a result of observation of videos, significant behavioral changes that we expected were not observed between pretest and posttest. However, some small changes reflected the debugging process they learned were observed [Observation Fact; OF] as below:

For Type A learners, it was observed that "[OFI-1] Frequencies of cursor moving that may be checking return value of function were increased (1 learner)" and "[OF I-2] Frequencies of cursor moving that may be checking branch statement were increased (1learner)." For Type B learners, it was observed that "[OF I-3] the learners who could not do anything after executing once in the pretest tried to watch the output of their modified program in the posttest. The statements the learners tried to modify were correct, however, they could not modify correctly (3 learners)."

Concerning Tobsync use, it was observed that all learners did synchronized observation by using next/prev buttons, (5) in Figure 3. Some learners temporary observed behavior of each program by using next/prev buttons, (3), (4) in Figure 3. However, they would back to use synchronize observation function [OF II-1]. Besides, no learner stopped their working for a long time [OF II-2].

5.3.2 Descriptions in questionnaire

Q1 is "Learning with this system is more understandable than ordinary classes" (Yes/No, Describe reason; for Hypothesis 1). Q2 is "Please describe the debugging process that you learned by our system" (Free describing; for Hypothesis 1). Only Type A learners answered Q1 and Q2 because these are for evaluation of LSDP-WS. Q3 is "Is it easier to observe the changes of value with Domain World than ordinary method?" (Evaluation with 1 to 5 points, Describe reasons; for Hypothesis 3). Q4 is "Is it useful to compare the behaviors of your program to ones of correct program to find inappropriate statement in your program?" (Yes/No, Describe reason if no; for Hypothesis 3)

For Q1, the average point is 3.0. The learner who gave high score (point 4 or 5) wrote the reason that "we can consider input and output for each function." The learner who gave low score (point 1 or 2) wrote the reason that "I didn't know what I was doing, but I thought it is important to write hypotheses." For Q2, learners wrote: "[D2-1] Writing functions' roles, arranging function relations, searching inappropriate point by checking input and output." and "[D2-2] Making structure chart of function relations and predicting inappropriate point from output, describing hypotheses from prediction, following program's behavior, and then find inappropriate point." Evaluation Score of Q3 [ES 3] is as follows: (1) the average points given by Type A is 5.0 and the average points given by Type B is 3.6. (2) Low score (point 1 or 2) was not marked. They gave reasons like: "[D3-1] I could compare my program's behavior to correct one, so that I wasn't in situation that I lose what I have to do", "[D3-2] It is easier to find mistakes by checking the difference between my program and correct program," and "I could watch program's behaviors in real time." For Q4, 9 learners answered "Yes" and 1 learner answered "No." The learner who answered "No" described reason, as "I am not good at programming."

5.3.3 Result of pretest and posttest

As for Type A learners, 1 learner could fix bug at the pretest and the posttest and that is same learner. As for Type B learners, no learner could fix the bug at the pretest. In contrast, 2 learners became to fix the bug at the posttest.

5.4 Consideration

As for our observation, learner's behavioral changes were small [OF I-1, I-2], and we couldn't observe significant behavioral changes. As for the descriptions in questionnaires, we can find some learners referred systematic debugging process partially [D2-1, D2-2]. These mean that some learners became aware of the systematic debugging process, however they could not follow the process correctly. Therefore, we suppose that hypothesis 1 was supported partially. We consider that only three problems were not enough to change the behavior of the learners.

We tried to confirm Hypothesis 2 (effectiveness of visualizing in Domain World) from Q3 and Hypothesis 3 (effectiveness of Tobsync) from Q4, but as for the Q3, we observed that they didn't answer about visualizing in Domain World and Tobsync individually. Some learners' answers at Q3 included their feels not on Domain World, but on Tobsync (see [D3-1], [D3-2]). For that reason, it is conceivable that [ES 3] reflects not only evaluation of Domain World, but also evaluation of Tobsync. Then, we evaluated Hypothesis 2 and 3 comprehensively from descriptions in questionnaire. From [ES 3] and [D3-1, D3-2], it can be said that synchronized observation in Domain World is evaluated favorably. As for the Q4, most of learners answered "Yes" so we can say that comparing learner's programs and correct programs by Tobsync is effective. From the observation of the video, it is conceivable that all learners compared their program and correct program [OF II-1]. Considering [OF I-3] as well, it is suggested that even the learners who could not do correct debugging process in the pretest could partially do themselves in the posttest. In addition, from [OF II-2], we could assume that comparison task with Tobsync is not difficult for learners. These support Hypothesis 3. In conclude, we observed some facts that support Hypothesis 3, but we are not sure that Hypothesis 2 was supported.

6. Conclusion

In this paper, we developed a learning environment on a systematic debugging process for novice learners. The learning environment has 2 types of important units: LSDP-WS and Tobsync. Then, we carried out a preliminary experiment to evaluate the effectiveness of our learning environment. The hypotheses on the effectiveness were partially supported.

In our future work, we will carry out another experiment to investigate novice learners' problems in debugging. Besides, it seems our current system's capability is limited. This is because we think comparing actual and expected output is important. We have implemented system for this factor first. Thus, we will expand assisting function suitably according to the results of evaluation experiment and the investigation.

Acknowledgements

This research is supported by Japanese Grant-in-Aid for Scientific Research (B) 24300282.

References

Egi, T., & Takeuchi, A. An Analysis on a Learning Support System for Tracing in Beginner's Debugging. (2007). *Proceedings of the 2007 conference on Supporting Learning Flow through Integrative Technologies,* 509-516.

Hanson, D,R., & Korn, J, L. (1997). A Simple and Extensible Graphical Debugger. *Proceedings of the USENIX 1997 Annual Technical Conference,* 174-183.

Imaizumi, T., Hashiura, H., Matsuura, S., & Komiya, A. (2010). A Programming Learning Emvironment "AZUR": Visualizing Block Structures and Program Function Behavior. *Proceedings of JCKBSE' 10,* 306-311.

Kogure, S., Okamoto, M., Noguchi, Y., Konishi, T., & Itoh, Y. (2012). Adapting Guidance and Externalization Support Features to Program and Algorithm Learning Support Environment. *Proceedings of International Conference on Computers in Education,* 321-323.

Moreno, A., Myller, N., & Sutinen, E. (2004). Visualizing programs with jeliot 3. *AVI 04: Proceedings of the working conference on Advanced visual interfaces,* 373-376.

Myers, G, J., Badgett, T., & Sandler, C. (2012). *The Art of Software Testing 3rd Edition.* New jersey, John Wiley & Sons, Inc.

Yokomori, R., Oohata, F., Takata, Y., Seki, H., & Inoue, K. (2001). Analysis and Implementation Method of Program to Detect Inappropriate Information Leak. Proceeding of The Second Asia-Pacific Conference on Quality Software, 5-12.