Shih, JL. et al. (Eds.) (2023). Proceedings of the 31st International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Program Visualization System Supporting Teacher-Intended Stepwise Refinement

Koichi YAMASHITA[a*], Hiroki SOMA[b], Satoru KOGURE[b], Yasuhiro NOGUCHI[b], Raiya YAMAMOTO[a], Tatsuhiro KONISHI[b] & Yukihiro ITOH[c]
[a]*Faculty of Business Administration, Tokoha University, Japan*
[b]*Faculty of Informatics, Shizuoka University, Japan*
[c]*Shizuoka University, Japan*
*yamasita@hm.tokoha-u.ac.jp

Abstract: This study portrays a learning support system that visually represents the structure of stepwise algorithm refinement, also known as the structure of program code abstraction, aligned with the teacher's instructional objectives. We focus on the varying needs of learners in programming education and consider the possibility that different levels of stepwise refinement may be required based on their knowledge and experience. Novice learners with limited programming experience may need program code abstractions in finer steps, rather than the abstraction of numerous statements simultaneously. For learners with a certain level of experience, abstraction with excessive steps may hinder their holistic understanding of the entire program from a higher perspective. In this study, we developed a program visualization system that empowers teachers to customize the visualizations of the structure of stepwise abstractions and the data structures utilized by the target program according to their instructional intentions. We introduced the proposed system into an actual class and evaluated its effectiveness by measuring the degree of improvement in learners' understanding based on tests.

Keywords: Programming education, program visualization system, stepwise refinement and abstraction, educational authoring tool

1. Introduction

To date, various program visualization (PV) systems have been developed to support the programming learning of novice learners (Sorva, Karavirta, & Malmi, 2013). We had developed a PV system called Teacher's Explaining Design Visualization Tool (TEDViT) and conducted several classroom sessions introducing this system into actual classes (Kogure et al., 2014). There are two features of TEDViT that distinguish it from other existing systems: first, it enables teachers to customize PVs based on their objectives, and second, it enables learners to observe and compare two PVs with different abstraction levels of target data structures. However, it cannot provide visualizations at different levels of abstraction for program-code that represents concrete operational procedures for the target domain world of the data structures to be processed.

In programming education, stepwise refinement has been used to design program codes by gradually decreasing the level of abstraction from highly abstract algorithms. Moreover, there have been various attempts to support learners' understanding of programs by focusing on stepwise abstraction used to understand the algorithm behind program code by gradually increasing the level of abstraction of the code. In a classic study, Shneiderman & Mayer (1979) summarized the procedures that expert programmers undertake during code reading. These steps include chunking some statements within the target program, recognizing the function of the chunk, and progressively assembling the chunks to form larger units. The chunking process is continued recursively until the entire program is understood. This implies that learners need to perform stepwise abstraction by chunking concrete operation procedures to understand a program. However, the literature suggests

that novice learners may frequently struggle to comprehend a program owing to difficulties in effectively chunking the code.

In this study, we developed a system that can visualize program behavior by drawing target program codes that represent concrete processing procedures and problem analysis diagrams (PADs) that represent algorithms with a higher level of abstraction than program code, enabling users to gradually change abstraction level of PADs. The abstraction level of PAD here refers to the size of blocks of program code to be chunked, with larger chunked blocks being considered highly abstract and smaller chunked blocks being considered less abstract. This system is an extension of TEDViT that enables teachers to define the chunks to be included in algorithm visualization. Learners understand chunks differently depending on their knowledge and experience. Hence, our system, which allows teachers to define chunking structures, is expected to provide PVs that better align with learners' understanding compared to existing systems with fixed chunking structures determined by developers.

2. Related Works

2.1 TEDViT

Price, Baecker, & Small (1998) categorized users of PV systems into the following four roles: *User/viewer* who observes the world visualized by the system; *visualizer/animator* who defines the visualization of the target program's behavior; *software developer* who develops the PV system; and *programmer* who designs the program that is the target of visualization. A typical existing PV system assigns the role of *user/viewer* to the learners, *visualizer/animator* and *software developer* to the PV system's developer, and *programmer* to the learners and teachers.

As mentioned earlier, we had developed a PV system called TEDViT, which assigns the role of *visualizer/animator* to the teachers. TEDViT is a system that enables teachers to define drawing rules to customize the visualization suitable for each target program and for the target user/viewer. When a teacher explains a program behavior to novice learners using visualizations, the teacher may sort out, highlight, and lay out some drawing objects from the target data structure. The teacher may also attach descriptive natural language texts to some objects to assist students' understanding and may draw pointers and array indexers using arrows. We consider these visualizations to be based on a policy that the teacher determines according to the learners' background knowledge and degree of understanding, the properties of the target program, etc. We refer to these visualization policies and the concepts that the teacher intends to explain using these policies as the teacher's "intent of instruction." TEDViT is, therefore, a PV system that can reflect teachers' intent of instruction regarding visualizations by assigning them the role of *visualizer/animator*.

TEDViT enables *visualizers/animators* to create or edit a configuration file for PV creation based on their intent of instruction independently from the target program file. Configuration file comprises a set of drawing rules, each of which is a comma-separated value (CSV) entry consisting of a condition part and an object part. The condition part defines the condition to fire the drawing rule. The object part defines the operation to edit the target object (i.e., "create," "delete," and "update" are available) and the attributes necessary to draw the object such as the object type, position, color, and corresponding variables. TEDViT generates PVs by scanning configuration files, interpreting the visualization policy represented by a set of drawing rules, and visualizing drawing objects accordingly. *Users/viewers* can observe the program behavior based on the PVs. The supporting programming language is C.

2.2 Programming Learning Support Based on Stepwise Refinement and Abstraction

In general, a PV system visualizes the target domain world involved in program processing and aims to support learners in understanding the program's meaning by enabling them to observe the changes in the domain world caused by the program. Each statement is used as

a unit of meaning, that is, a unit that causes changes in the domain world. Although PV systems can visually support the understanding of the meaning of each statement, they may not be able to provide sufficient support for abstracting the meaning of multiple statements and understanding it, and hence, understanding the meaning of the entire program.

The stepwise refinement technique has been widely used in programming education. The procedure summarized by Shneiderman & Mayer (1979) implies that program comprehension can be achieved by the opposite operation, namely, stepwise abstraction from the detailed meaning of individual statements to abstract meaning that spans multiple statements. Here, abstraction is the interpretation of the meaning of chunking and chunks of the program, and refinement is the decomposition of chunks into smaller chunks. In learning programming, it is also necessary to understand the meaning of programs through the abstraction of operational procedures and refinement of algorithms; this is not supported by PV systems. Based on this idea, there have been several attempts to aid novice programming learners by supporting the stepwise abstraction of programs and pseudocode (Shinkai & Sumitani, 2007; Watanabe, Tomoto, & Akakura, 2015; Kogure et al., 2013).

3. Proposed System

In this study, we focused on the possibility that the learners may need different granularities of chunking in programming learning based on stepwise refinement depending on their knowledge and experience. For novice learners with minimal programming experience, small chunks of program code have to be abstracted at multiple levels of abstraction, rather than chunking a large amount of code simultaneously. For learners with a certain level of experience, chunking extremely granularly may hinder their holistic understanding of the entire program from a higher perspective. Yamashita et al. (2016) highlighted that there are two possible types of abstraction of sequences of operations: simple grouping and process generalization. In our teaching experience, learners with insufficient knowledge of CS and mathematics often struggle to understand chunking, particularly when it involves process generalization. In such cases, teachers could customize the chunking granularity and explanation of the meaning of the chunks based on the differences in learners' knowledge and background. Moreover, the differences in the chunking granularities would have a certain effect on their visualization policy of the target domain world.

In this study, we developed a PV system that enables teachers to freely define and visualize the chunks of program code and their structures based on their intent of instruction. The chunks of program code are visualized as PAD-based algorithmic representations. Our system can enable users to increase the abstraction granularity of each chunk object and visualize more detailed PADs. It builds upon TEDViT, a program visualization (PV) system that empowers teachers to customize the visual representations of the domain world according to their instructional objectives. Therefore, our system provides teachers with the flexibility to define and customize the visualizations of the data structures involved in program processing. Figure 1 provides a screenshot of the proposed system.

Our system generates a learning environment with four main visualization areas. In area (A), the target program code is visualized, and an executing statement or chunk is highlighted. In area (B), a PAD-based algorithm representation is visualized. Area (C) visualizes the currently executed operation, displaying the text of the executing chunk. Area (D) visualizes the domain world, drawing the data structure of the program's processing target based on the definition file edited by the teacher as in TEDViT.

Each element of the PAD in area (B) corresponds to an abstracted chunk of statements in a certain range of the program code. The correspondence between each element of the PAD and each range of the program code can be identified by the number assigned to each element of the PAD and each range of the program code. In addition, buttons such as "+" and "-" are provided at the top of each element of the PAD. By clicking these buttons, the chunk of each element can be displayed in one more step in detail, as shown in Figure 2. As the algorithm representation visualized in area (B) includes chunking structures, it cannot strictly be called a PAD. We call the algorithm representation visualized

by the proposed system "extended PAD." In addition to the aforementioned features, some elements of the extended PAD have buttons for "step in function" and "step over function," resembling typical debuggers. Some elements also have buttons for "execute all loop" and "execute one round of loop" depending on the processing contents.

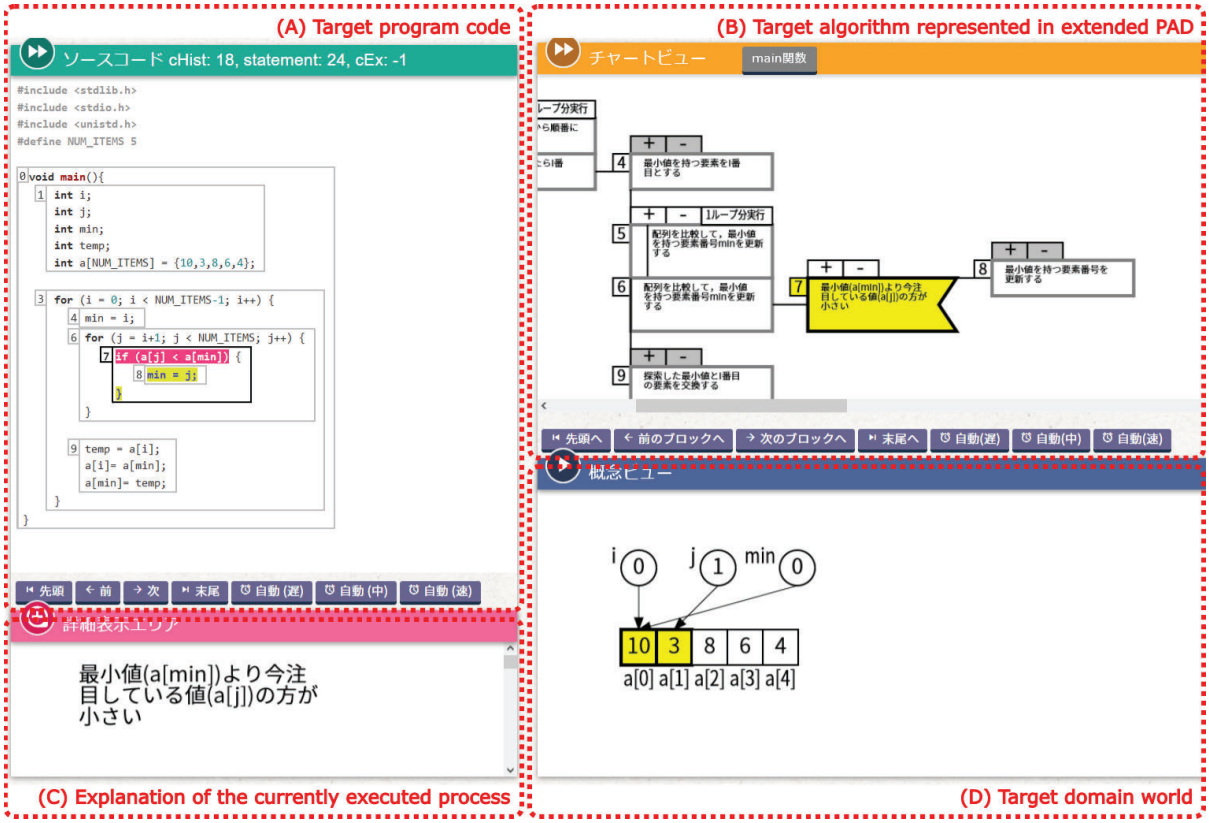

Figure 1. Screenshot of the proposed system.

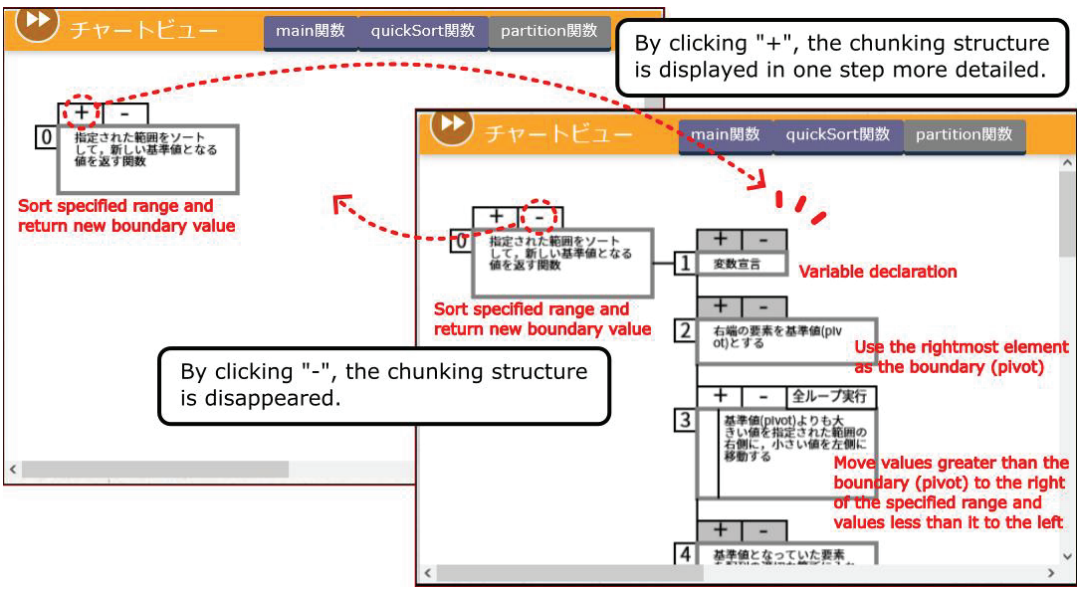

Figure 2. Visualizing a chunking structure in one step more detailed.

4. Evaluation

To assess the effectiveness of our system, we conducted a classroom practice by incorporating it into an actual programming course. The hypothesis to be verified is whether

our system could help learners to understand the behavior of programs and algorithms in code reading learning better than an existing PV system, TEDViT. The actual class whereby the systems were introduced was an on-demand class offered to second-year university students majoring in computer science. The number of participants in our class was 47, and they had 1-2 years of programming experience. Two types of sample programs, quick sort and merge sort, were prepared for code reading. All participants had already learned quick sort in the previous semester but had not learned merge sort.

In the practiced class, we had the participants perform code reading learning in the order of quick sort first and merge sort second. In each code reading learning, they took a pre-test before learning with PV systems, and then performed two learnings of code readings, one with TEDViT and the other with our system. They took tests immediately after each of the two learnings, which were positioned as a mid-test and a post-test, respectively. To reduce order effects, we divided the participants into two groups, Groups A and B, with each group learning with PV systems in reversed order. We also reversed the order of use of the system in the two learnings of quick sort and merge sort for each group as well. After all the learnings were conducted, we conducted a questionnaire survey.

In the code reading learning using TEDViT, participants observed changes in the logical data structure visualized in the domain world with stepwise execution of each statement of the target program. In the learning using our system, participants used the "+" and "-" buttons provided in area (B) of Figure 1 to observe the chunking structure of the extended PAD and simultaneously observed the changes in the domain world with stepwise execution of each statement or each chunk. The pre-test, mid-test, and post-test consisted of nine questions for the quick sort and 11 questions for the merge sort and were identical for each target program. Participants were required to take six tests in the entire class, and a total of 32 participants (13 in Group A and 19 in Group B) took all six tests.

Table 1 summarizes the average correct answer rates for each test for the 32 participants who took all tests. Underlined scores are the rates for the tests immediately after learning with our system. Both Groups A and B, regardless of the target program, demonstrated improved scores after code reading learning compared to before learning, suggesting that participants would improve their understanding of the programs. We also calculated average improvements for learning with TEDViT and for learning with our system by calculating score differences for all test results compared to the results of the immediately preceding test. We found the average improvement of correct answer rates for learning with TEDViT was 9.2% and that for learning with our system was 14.4%. Currently, we have not obtained any statistical results that significantly support our hypothesis that our system contributes more to learners' understanding of programs than TEDViT. However, the observed average improvement of correct answer rates for our system, which is 5% higher than that of TEDViT, suggests a significant degree of effectiveness in enhancing learners' understanding.

Table 1. *Average correct answer rate of pre-, mid-, and post-test*

Group	N.	Target	Pre-test	Mid-test	Post-test
A	13	Quick sort	.554	.662	<u>.708</u>
A	13	Merge sort	.608	<u>.746</u>	.738
B	19	Quick sort	.332	<u>.668</u>	.679
B	19	Merge sort	.526	.768	<u>.758</u>

The post-class questionnaire included a 5-point pairwise comparison of whether TEDViT or our system was easier to learn, with 1 indicating a strong preference for TEDViT, 5 indicating a strong preference for our system, and the median being 3. The mean value for this item was 3.3 for the 30 participants who responded to the questionnaire, indicating that participants tended to prefer our system to TEDViT. Moreover, the mean values of all the items for which participants were asked to subjectively evaluate the effectiveness of the functions of our system were also higher than the median. Subjective evaluations of the

learning effectiveness of our system were also favorable, suggesting that participants positively viewed our system.

5. Conclusion

This study described a PV system that can simultaneously provide learners with processing procedures and processing objects at different levels of abstraction. The system achieves this by generating visualizations of the stepwise chunking structure of program code and visualizations of logical data structure, which are the objects being processed. We assessed the effectiveness of our system by introducing it in an actual class. As our system is based on the existing PV system, TEDViT, we asked participants to perform code reading learning using TEDViT and learning using our system and measured the effects of two learning activities by administering pre-, mid-, and post-tests. To reduce order effects, participants in the practice class were divided into two groups, each with an opposite learning sequence. In addition, two target programs for code reading, quick sort, and merge sort, were prepared, and the effect of the systems was measured on multiple program codes. The evaluation results based on the correct answer rates in three tests indicated no statistical difference between code reading using TEDViT and code reading using our system. However, TEDViT achieved a 9.2% improvement in correct answer rate on average, while our system achieved a 14.4% improvement in correct answer rate on average, indicating that the improvement was more than 5% better for learning with our system. We can conclude that the results indicate a certain degree of support for the effectiveness of our system.

Acknowledgements

This study was supported by JSPS KAKENHI Grant Numbers JP19K12259 and JP22K12290.

References

Kogure, S., Fujioka, R., Noguchi, Y., Yamashita, K., Konishi, T., Itoh, Y. (2014). Code reading environment according to visualizing both variable's memory image and target world's status. *Proceeding of the 22nd International Conference on Computers in Education*, 343-348.

Kogure, S., Okamoto, M., Yamashita, K., Noguchi, Y., Konishi, T., Itoh, Y. (2013). Evaluation of an algorithm and programming learning support features to program and algorithm learning support environment. *Proceedings of the 21st International Conference on Computers in Education*, 418-424.

Price, B., Baecker, R., Small, I. (1998). An Introduction to software visualization. In J. Stasko, J. Domingue, M. H. Brown, & B. A. Price (Eds.), *Software Visualization* (pp. 3-27). MIT Press.

Shinkai, J., Sumitani, S. (2007). Development of programming learning support system emphasizing process. *Japan Journal of Educational Technology*, *31*(Suppl.), 45-48.

Shneiderman, B., Mayer, R. (1979). Syntactic/Semantic interaction in programmer behavior: A model and experimental results. *International Journal of Computer and Information Sciences*, *8*(3), 219-238.

Sorva, J., Karavirta, V., Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, *13*(4), 15.

Watanabe, K., Tomoto, T., Akakura, T. (2015). Development of a learning support system for reading source code by stepwise abstraction. In S. Yamamoto (Ed.), Human Interface and the Management of Information. Information and Knowledge in Context. HIMI 2015. *Lecture Notes in Computer Science* (Vol. 9173, pp. 387-394). Springer, Cham.

Yamashita, K., Nagao, T., Kogure, S., Noguchi, Y., Konishi, T., Itoh, Y. (2016). Code-reading support environment visualizing three fields and educational practice to understand nested loops. *Research and Practice in Technology Enhanced Learning*, *11*(3), 1-22. doi:10.1186/s41039-016-0027-3