Kashihara, A. et al. (Eds.) (2024). Proceedings of the 32nd International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Program Learning Support System with Visualization Reflecting Teacher's Intent for Learner's Code

Kenzo KOBAYASHI[a*], Satoru KOGURE[b], Yasuhiro NOGUCHI[b], Raiya YAMAMOTO[c], Koichi YAMASHITA[c], Tatsuhiro KONISHI[b] & Yukihiro ITOH[d]

[a]Graduate School of Integrated 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
*kobayashi.kenzo.19@shizuoka.ac.jp

Abstract: Understanding program behavior is challenging for novice learners. To address this problem, various program visualization systems such as TEDViT have been developed. TEDViT allows teachers to establish drawing rules, ensuring that their intentions are reflected in the visualization results. However, visualizing programs edited by learners using TEDViT has been hindered by its rule notation. In this study, we propose a novel rule designed to accommodate learner-edited programs. Furthermore, we develop a new system based on this rule.

Keywords: programming, learning support system, visualization

1. Introduction

Understanding a program's behavior can be challenging for novice learners. Program visualization has proven to be effective in assisting learners (Mitsui & Nakamura, 1994), leading to the development of various systems such as Jeliot 3 (Moreno et al., 2004) and JAVAVIS (Oechsle & Schmitt , 2002) for visualizing program behavior. However, these systems cannot reflect a teacher's intent in the visualization results, as they typically perform predefined visualizations based on developer-set rules. When introducing such systems in classrooms, the visualization results should align with the teacher's explanations. Teachers can provide better lectures with the system's visualization results if they can arrange and emphasize objects, and provide supplementary explanations according to their own intentions.

TEDViT (Yamashita et al., 2017) was developed to generate visualizations aligned with teachers' intentions. Through TEDViT, teachers can create drawing rules, as described below, ensuring that their intentions are reflected in the visualization results. The usefulness of TEDViT has been demonstrated through experiments involving real learners (Kogure et al., 2014). However, because TEDViT was initially developed to visualize programs created solely by teachers, it encountered challenges in visualizing programs edited by learners. In this study, we developed a system that reflects teachers' intentions and visualizes the programs edited by learners.

2. Related Research

Sorva et al. (2013) proposed the 2 Dimension Engagement Taxonomy (2DET) for classifying and assessing program visualization systems. 2DET evaluates the learning effectiveness of

each program visualization system by considering two independent indicators: direct engagement and content ownership.

Direct engagement classifies systems into seven levels based on their functionality. The following four of the seven 2DET levels were used in this study: No Viewing: Systems do not provide visualization; Viewing: Learners can view the visualization; Controlled Viewing: Learners can control how they view visualizations, for instance, by changing the animation speed; and Creating: The learner creates a novel way of visualizing the target programs.

Content ownership is divided into the following four levels determined by the degree of learners' capability to modify the program code: Given Content: Learners simply observe the provided program code; Own Cases: Learners can define the input or other parameters of their program code; Modified Content: Learners can modify their program code; and Own Content: Learners view visualizations of the program code they wrote themselves.

Systems where direct engagement is classified as controlled viewing or lower include CSmart (Gajraj et al., 2011) for visualizing given content, metaphor-based OO visualizer (Sajaniemi et al., 2006, 2007) for visualizing Own Cases, and Jeliot 3 and JAVAVIS for visualizing Own Content. These systems have achieved stable visualization for any program code. This is because learners do not have the capability to influence visualization content. Because the systems only need to draw objects according to the variable data of their target program, developers can design visualization policies in advance, regardless of the target program's content or its internal state.

In contrast, in the case of Creating, this level involves devising a "novel way to visualize the target programs," where systems classified as Creating are expected to visualize the target program in accordance with its contents. This includes altering the visualization policy depending on the content of the target program and its internal state during execution. Animal (Rößling & Freisleben, 2002) and TEDViT allow users to adjust their visualization policies. However, Animal is classified as given content and TEDVIT falls into the category of Own Cases in terms of content ownership. This is because in these systems, the most flexible approach to altering the visualization policy, depending on the content of the target program and its internal state during execution, specifies the step number during program execution, for instance, a directive such as "make the color of object A red in step 5." Because the code structure, such as variable names, step numbers, and line numbers, may be changed by the learners' modification, this method may not be sufficient to achieve adequate visualization of programs edited by learners.

3. Program Learning Support System with Visualization Reflecting Teacher's Intent for Learner's Code

When attempting both Creating and Own Content, it becomes challenging to anticipate the code's structure when writing rules. Therefore, this study aims to provide support for both Creating and Own Content with certain restrictions on content ownership. These restrictions must be maintained to the minimum necessary to provide the support for Creating. To this end, we introduce a new system, referred to as MC-TEDViT, based on the TEDViT concept.

In TEDViT, users customize the visualization policy by writing the drawing rules. Users can draw variable objects and explanatory auxiliary objects such as arrows and comments. A drawing rule comprises a core component and an activation condition. The core component determines the color, shape, coordinates, etc. of the object, and the activation condition determines the step at which each drawing rule is to be activated. Because the activation conditions are important for discussing the classification of TEDViT within 2DET, this section discusses the activation conditions.

The activation conditions of TEDViT faces two problems: Problem 1: Certain conditions depend on the step and line numbers; Problem 2: These conditions are insufficient for uniquely identifying lines with identical content. To address Problem 1, we have added several activation conditions to MC-TEDViT, for instance, "entry('funcA')" activates the drawing rule when a method named "funcA" is called during execution. This condition does not depend on either the line or step number. To address Problem 2, we have enabled logical AND and logical

OR to be implemented, for instance, an activation condition “inMethod(‘funcA’) && declared(‘x’)” activates the drawing rule only when a variable named “x” is declared in the scope of “funcA” even if the target program has an identical line in the scope of “funcB.” These activation conditions serve to mitigate problems and specify the typical steps in program execution used by teachers in their drawing rules for TEDViT.

In terms of implementation, MC-TEDViT has been developed as an extension for Visual Studio Code. Data collection is accomplished through the Java Debugging Interface, an API designed for debugging and static analysis of the program. The drawing rules are written in TOML format.

To verify the effectiveness of the proposed method, we conducted a simple experiment with MC-TEDViT in its development phase, which suggested that Problems 1 and 2 could be solved.

4. Conclusion

In this study, MC-TEDViT was developed with the aim of constructing a programming learning system that not only “reflects the teacher’s intent” but “visualizes the learner’s edited program.”

Finally, we describe the current progress and the future. Currently, MC-TEDViT can analyze and visualize Java programs using primitive-type variables. However, it cannot handle class instances. This functionality is currently being developed and will be implemented in the near future. Subsequently, we intend to evaluate MC-TEDViT through an experimental evaluation scheduled in December 2024. This evaluation will involve first-year undergraduate students enrolled in the Department of Computer Science.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Numbers JP19K12259, JP22K12290, JP24K15219.

References

Gajraj, R. R., Williams, M., Bernard, M., & Lenandlar S. (2011). Transforming Source Code Examples into Programming Tutorials. *Proceedings of ICCGI'11*. 160-164.

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. *Proceedings of ICCE2014, 343-348.*

Mitsui, K., & Nakamura, H. (1994). Visualization Techniques for Understanding Object-Oriented Programs. IPSJ SIG Software Engineering (SIGSE), 55, 129-136.

Moreno, A., Myller, N., & Sutinen, E. (2004). Visualizing programs with Jeliot 3. Proceedings of *AVI '04* (pp. 373-376). ACM Press.

Oechsle, R., & Schmitt, T. (2002). JAVAVIS: Automatic program visualization with object and sequence diagrams using the Java Debug Interface (JDI). In Revised Lectures on Software Visualization, International Seminar, S. Diehl Ed.

Rößling, G. & Freisleben, B. (2002). ANIMAL: A System for Supporting Multiple Roles in Algorithm Animation. *Journal of Visual Languages and Computing*, *13*(3), 341-354.

Sajaniemi, J., Byckling, P., & Gerdt, P. (2006). Metaphor-based animation of OO programs. Proceedings of SoftVis’06. 173-174.

Sajaniemi, J., Byckling, P., & Gerdt, P. (2007). Animation Metaphors for Object-Oriented Concepts. *Electronic Notes in Theoretical Computer Science*, *178*, 15-22.

Sorva, J., Karavirta, V., & Malmi, L. (2013). A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Transactions on Computing Education*, *13*(4), 1-64.

Yamashita, K., Fujioka, R., Kogure, S., Noguchi, Y., Konishi, T., & Itoh, Y. (2017). Classroom Practice for Understanding Pointers using Learning Support System for Visualizing Memory Image and Target Domain World. *Research and Practice in Technology Enhanced Learning*, *12*(17), 1-16.