Yang, J. C. et al. (Eds.) (2018). Proceedings of the 26th International Conference on Computers in Education. Philippines: Asia-Pacific Society for Computers in Education

# A Learning Support System for Visualizing Behaviors of Students' Programs Based on Teachers' Intents of Instruction

**Koichi YAMASHITA[a*], Daiki TEZUKA[b], Satoru KOGURE[c], Yasuhiro NOGUCHI[c], Tatsuhiro KONISHI[c] & Yukihiro ITOH[c]**
[a]*Faculty of Business Administration, Tokoha University, Japan*
[b]*Graduate School of Integrated Science and Technology, Shizuoka University, Japan*
[c]*Faculty of Informatics, Shizuoka University, Japan*
*yamasita@hm.tokoha-u.ac.jp

**Abstract:** In this paper, we describe a learning support system for visualizing the behaviors of students' programs and discuss classroom practices utilizing the system. Up to present, we have held several classes using a program visualization system called TEDViT, which visualizes the behaviors of teachers' programs based on teachers' intents of instruction. However, TEDViT does not support visualization of students' own programs; hence, learners can only observe the behaviors of a given teacher's program. In this study, we extended TEDViT to be capable of visualizing learners' own programs in a drawing style that reflects the teacher's intent of instruction. We introduced the extended TEDViT into three classes and evaluated the robustness and usefulness of the visualizations of learners' programs. We designed an improvement for our extended TEDViT based on knowledge derived from these practices. The evaluation results suggested that the extended TEDViT's visualizations have a certain degree of robustness and that our approach has some validity.

**Keywords:** Programming education, program visualization system, domain world model, classroom practice

## 1. Introduction

Thus far, several program visualization (PV) systems have been developed to support novices who are learning programming (Sorva, Karavirta & Malmi, 2013). These systems visualize the data structures processed by the target programs (i.e. target domain world) in a uniform way and help learners to understand the targets by making their behavior visible. We adapted a PV system called the Teacher's Explaining Design Visualization Tool (TEDViT) (Kogure et al., 2014) and held several classes using it over the past few years (Yamashita et al., 2016a; Yamashita et al., 2017). TEDViT allows teachers to provide not only a target program, but also its visualization policy. The visualization policy is defined by a set of drawing rules, each of which consists of a condition part representing the drawing timing and an object part representing the object's attributes, such as type, position, and color. The teacher can define what objects are visualized and when they are visualized in the process of program execution by providing the target program and drawing rules to TEDViT. However, TEDViT does not support visualizations of learners' own programs. This is mainly because managing the timing to fire drawing rules is difficult.

Therefore, the aim of this study is to extend TEDViT to be capable of visualizing learners' own programs in a drawing style that reflects the teacher's intent of instruction. In our extension, we take an approach that expresses the condition parts of drawing rules with superficial statuses in the program's execution, because the functions of novices' statements or program code blocks are hard to analyze automatically. We introduce the extended TEDViT into actual classes and evaluate the robustness and usefulness of the visualizations of learners' programs. Through repeated use in classrooms, we attempt to develop a highly sustainable PV system for use in actual classes. In this paper, we describe the extended TEDViT that supports visualizations of learners' own programs and our three classroom practices with it.

## 2. Visualization of Student's Own Program Behavior

TEDViT allows teachers to define the policy for drawing a status of the target domain world based on their intents of instruction. Teachers can create or edit a configuration file independently from the target program file. TEDViT interprets such a visualization policy by scanning the configuration file and then visualizes the target domain world accordingly. The learners can then observe the program behavior in the target world visualized in accordance with the teacher's intent of instruction. The relationships among teachers, learners, and TEDViT, and the extension in this study, are shown in Figure 1. A configuration file that defines a visualization policy consists of a set of drawing rules, each of which is a CSV (Comma-Separated Values) entry consisting of a condition part and an object part. A condition part defines the condition to fire the drawing rule. Teachers can express the timing of drawing using a conditional equation consisting of a statement ID, variables in the target program, constant values, and comparison operators. Here, the statement ID is a unique identifier automatically assigned to all statements in the target program by TEDViT. An object part defines the operation to edit the target object and the attributes necessary to draw the object.

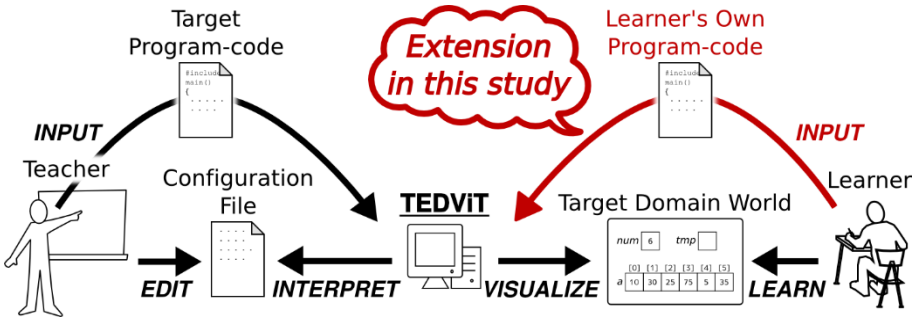


*Figure 1.* Relationships among teachers, learners, and TEDViT

In our past practical classes with TEDViT, the teachers defined a total of 725 drawing rules for 23 target programs, and all of the rules contained conditional equations with a statement ID. This was because the drawing rules had to be efficacious only for the target program provided by the teacher. Using a statement ID in a condition part means defining a condition such as "when *n*-th statement in the target program is executed." If the target program were the learner's own program, the order and position of the statements would be different according to the learner's program, even if the learners' programs achieved the same result. Thus, when TEDViT visualizes the behaviors of learners' programs, conditions with statement ID cannot fire the drawing rule at appropriate time.

To resolve the fundamental problem in drawing management for learners' code, we thought it necessary to extend TEDViT to support expressing condition parts with statuses abstracted to the level of functions, such as "when the indexer of the target array points out of the array range," rather than concrete and superficial statuses, such as "when *n*-th statement is executed." However, we thought this would be hard to implement. For the abstract drawing management, analyzing the provided learner's program and finding the functions of each code block and data structure are required. Even expert teachers find it hard to interpret abstract functions in novice learners' programs because there are many redundancies and errors. Therefore, in this study, we addressed drawing management for learners' code by a condition part expressed with concrete and superficial statuses, based on two approaches.

First, to identify the learner's statement that achieves the same process as the teacher's statement, we extended TEDViT to support following expressions in condition parts: "onChange(*var*)" is used to fire the rule when the value of variable *var* is changed by the statement execution. "cond(*cond*)" is used when conditional equation *cond* is satisfied. Here, the *cond* is expressed with variables in the target program and/or constant value. Logical operators including "AND" and "OR" are supported. "perfect(*str*)" and "partial(*str*)" are used when the expression of the executing statement matches the string *str* perfectly and partially, respectively. "regexp(*ptn*)" is used when the expression of the executing statement matches regular expression pattern *ptn*.

Second, we placed a restriction on designing exercise questions, making the students complete the program from the provided template that included definitions of the required variables, rather than making them develop it from full scratch. In full scratch development, students use an

arbitrary number of variables and arbitrary variable names to achieve the target algorithms. Hence, analyzing correspondences of variables in students' programs to variables in teachers' programs is required. By providing a template, we aimed to fix the number and names of variables, which we expected to facilitate the production of condition expressions with concrete and superficial statuses.

We believe the restriction on design exercises based on a completion strategy had very little influence on teachers' actual classroom designs. Van Merrienboer and Krammer (1987) showed that learners who learned programming using exercises based on a completion strategy came to develop better programs than learners who learned using exercises from scratch. We thought that it would involve little cost to change an exercise from a scratch into a completion problem by removing some parts of the worked-out programs.

## 3. Classroom Practices

Based on the approaches described in the previous section, it is practically difficult to achieve fully robust visualizations for fully arbitrary target programs. An implementation can be as redundant as one likes. Nevertheless, in actual programming exercises, it is very unlikely that novice learners will make extremely redundant programs aiming to hurt the robustness of the system's visualizations. This means that visualization robustness needs to be evaluated only for learners' programs written in actual classes, rather than for fully arbitrary programs. Therefore, we introduced the extended TEDViT into actual classrooms and evaluated its visualization robustness. We also evaluated the validity of our approaches based on knowledge derived from practice. In this section, we describe three practical classes for university students. Table 1 presents a summary of the classes.

Table 1

*Summary of the Practical Classes*

| | Class #1 | Class #2 | Class #3 |
|---|---|---|---|
| No. of participants | 59 | 108 | 117 |
| Major | Business administration | Computer science | Computer science |
| Grade | Sophomore | Freshman | Sophomore |
| Course | Programming | Algorithms & data structures I | Algorithms & data structures II |
| Material | String matching | Sorting algorithms | Merge sort |

### *3.1 Class #1 (String Matching) and Class #2 (Sorting Algorithms)*

Class #1 took place in 2017, incorporated into an actual course called "Programming." Before the students worked on the exercise, they heard some instructions on using the extended TEDViT and explanations of the exercise problem. The exercise problem was to complete a program to judge whether a given string is a palindrome or not. Before the class, the teacher made the answer program and defined its visualization policy in the extended TEDViT. He also made the program template by removing some parts of the answer program. We added a function to hide some parts of the program visualized in the extended TEDViT, because the teacher asked that we allow the students to compare two visualizations; one visualization for the behaviors of students' programs and one for the behavior of the answer program. This function enabled the extended TEDViT to visualize the answer program's behavior without presenting the completed answer program code. During Class #1, the teacher instructed the students to launch the two processes of the extended TEDViT simultaneously, one for the students' program and one for the answer program. He guided the students to compare the two programs' behaviors and make the behavior of their programs closer to that of the answer program.

Class #2 took place in 2017, incorporated into an actual course called "Algorithms and Data Structures I." Like in Class #1, the students received some instructions on using the extended TEDViT and explanations of the exercise problem, and then they worked on two exercises. The first exercise was to consider the features of the insertion sort algorithm and the quicksort algorithm. The students were instructed to achieve the objective by observing the behaviors of two programs

visualized in the extended TEDViT, changing the initial arrays of both programs to the indicated three values. The second problem, which was a challenging exercise for students who had finished the first exercise, was to upgrade the quicksort program provided in the first exercise. The provided program always chose the last element of the target array as a pivot. The students were required to implement a program block that chose the element with the second highest (lowest) value among the first, last, and center element of the target array. They were also required to consider the effectiveness of the upgrade to pivot choosing by observing the behavior of the upgraded program visualized by the extended TEDViT.

At the end of each class, we conducted questionnaire surveys to evaluate the robustness and usefulness of the visualization by the extended TEDViT. Both questionnaires had two questions that used a five-point scale; they asked "(Q1) how appropriately did the extended TEDViT visualize the behaviors of your programs?" and "(Q2) how much do you want to use the extended TEDViT in actual classes?" They also had another question with a free description (Q3), which asked the reason for the answer to Q2. Table 2 provides a summary of the answers to these questions. For the former two questions in Class #1, the students who answered positively (a score of 4 or 5) accounted for 49.2% for Q1 and 54.3% for Q2, while the ones who answered negatively (score of 1 or 2) accounted for 15.3% for Q1 and 13.6% for Q2. For Class #2, the ones who answered positively accounted for 89.7% for Q1 and 84.6% for Q2, while the ones who answered negatively accounted for 5.1% for Q1 and 1.3% for Q2. These results suggest that the extended TEDViT could visualize the behaviors of students' programs robustly, and that the students accepted the visualization favorably, in general.

Table 2

*Results of the Questionnaire Survey for Class #1 and Class #2*

| Score | N in Q1 (Class #1) | N in Q2 (Class #1) | N in Q1 (Class #2) | N in Q2 (Class #2) |
|---|---|---|---|---|
| 1 | 0 | 2 | 2 | 1 |
| 2 | 9 | 6 | 2 | 0 |
| 3 | 21 | 18 | 4 | 11 |
| 4 | 20 | 25 | 13 | 39 |
| 5 | 9 | 7 | 57 | 27 |

For Q3, we found that the numbers of students who answered "understandable/imaginable" or "difficult/incomprehensible" were relatively high in Class #1. The former positive answers suggest that the extended TEDViT could have a certain degree of usefulness. We consider the latter negative answers as being caused by the complicated operations for launching the extend TEDViT. The students needed to use some CUI operations to launch it, followed by every compilation of their own program, with which students majoring in business administration are unfamiliar. For Q3 in Class #2, we found that 35 students answered "understandable," suggesting that the extended TEDViT could have a certain degree of usefulness. Moreover, none of the students answered "difficult/incomprehensible" in Class #2. We consider the reason to be that the participants were students majoring in computer science, so they were familiar with the CUI operations.

In Class #2, we collected programs from the students who attempted the second exercise. After the class, the teaching staff evaluated the robustness of the visualizations by verifying those of the collected programs' behaviors visualized with the extended TEDViT. The number of students who performed the second exercise was five. We collected 3 programs from each student, for a total of 15. The visualization of each program was evaluated in three grades: "no error" if the visualization was equivalent to that of the teacher's program; "trivial error" if the visualization had some errors such that a drawing object was not deleted appropriately, which did not affect the understanding of the program's behavior; and "fatal error," if the visualization had some errors such as errors in drawing position and highlighting, which affected the understanding of the program's behavior. The evaluation results showed that no visualization was graded "no error," 5 were "trivial error," and 10 were "fatal error." The reasons for these unsatisfactory results could be roughly classified into two factors: there were some bugs in the extended TEDViT or there were some bugs in the drawing rules defined by the teacher. The former bugs were relatively uncomplicated, and we fixed them. The latter bugs were in condition parts written with regular expressions, and arose from the teacher's insufficient anticipation of the students' programs' variation. For example, although

the teacher anticipated the use of "return" in the void function, many students did not use it in their programs. If we fixed the regular expressions appropriately, all visualizations of the collected programs could be graded as "no error."

### *3.2 Class #3: Merge Sort*

Class #3 also took place in 2017, incorporated into an actual course called "Algorithms and Data Structures II." Like in Classes #1 and #2, the students heard some instructions for using the extended TEDViT and explanations of the exercise problem, and then they worked on four exercises. The first exercise (Ex1) entailed completing the provided template of a merge sort program using the extended TEDViT. The template was made by removing part (about 10 lines) of the answer program written by the teacher beforehand. The second and third ones (Ex2 and Ex3) involved considering the features of the merge sort algorithm by observing the behaviors of the program visualized in the extended TEDViT and changing the initial array to the indicated one for each exercise. The fourth one (Ex4) asked them to implement two functions, *divide*(), which divides the target array, and *merge*(), which merges the two targets, using the dividing function *mergesort*() in the provided program. Before the class, the teacher made the target program for observation of behavior and defined its visualization policy in the extended TEDViT, anticipating the students' program modifications in the second exercise. During the class, the teacher instructed the students not to describe statements such as *typedef*, which was not supported by the extended TEDViT.

In Class #3, we conducted objective evaluations as in Class #2. That is, we collected the students' programs for each exercise, then the teaching staff verified the visualizations by the extended TEDViT and evaluated their robustness. During program collection, we classified a 20% random sample of all 117 students and collected the programs of 23 sample students, which were submitted as products. Then, we excluded unfinished programs from the collection, such as those with compilation errors, and obtained a total of 81 students' programs. Likewise, in Class #2, the teaching staff evaluated each program's visualization by the extended TEDViT using three grades. Table 3 shows the number of evaluated visualizations in each grade for each exercise.

Table 3

*The Number of Evaluated Visualizations of Students' Programs in Class #3*

| | Ex1 | Ex2 | Ex3 | Ex4 |
|---|---|---|---|---|
| No error | 13 | 13 | 13 | 11 |
| Trivial error | 0 | 0 | 0 | 1 |
| Fatal error | 8 | 8 | 9 | 5 |
| Total | 21 | 21 | 22 | 17 |

The reasons for the "fatal error" grade could be classified roughly into two factors again: one was unsupported statements written by some students ignoring the teacher's instructions and the other was uncomplicated bugs in the extended TEDViT. For the former, we translated unsupported statements in the students' programs into supported statements that had the equivalent function, and verified the visualizations again. We also re-verified the visualizations after the bug fixes for the extended TEDViT. Table 4 provides these reevaluation results.

Table 4

*The Number of Reevaluated Visualizations with Grammatical Modifications and bug fixes*

*With grammatical modifications*

| | Ex1 | Ex2 | Ex3 | Ex4 |
|---|---|---|---|---|
| No error | 19 | 19 | 20 | 14 |
| Trivial error | 0 | 0 | 0 | 1 |
| Fatal error | 2 | 2 | 2 | 2 |
| Total | 21 | 21 | 22 | 17 |

*With grammatical modifications and bug fixes*

| | Ex1 | Ex2 | Ex3 | Ex4 |
|---|---|---|---|---|
| No error | 21 | 21 | 22 | 16 |
| Trivial error | 0 | 0 | 0 | 1 |
| Fatal error | 0 | 0 | 0 | 0 |
| Total | 21 | 21 | 22 | 17 |

From Table 3, we can see about 60% of students' programs could be visualized appropriately even if no modifications were made. Furthermore, as seen from Table 4, almost all students' programs could be visualized appropriately following our trivial modifications. These results suggest that the visualizations of the extended TEDViT had a certain degree of robustness.

## 4. Conclusion

In this paper, we described an approach to extend TEDViT so that it could visualize the behaviors of learners' programs. TEDViT is a PV system that allows teachers to provide not only a target program but also its visualization policy. This feature aims to reflect teachers' intents of instruction regarding the visualization of the target program's behaviors. Because it was assumed that the target program would always be provided by the teacher, TEDViT was not made to support visualizations of the behaviors of learners' programs. To enable learners to observe the behaviors of their own programs visually, we extended TEDViT to use concrete and superficial statuses of a program's execution in drawing rule definitions. It was difficult to make our extended TEDViT fully robust for visualizations of fully arbitrary target programs. Hence, we evaluated the visualization robustness using learners' programs written in actual classes. We described three classroom sessions where we introduced the extended TEDViT for evaluations of robustness. In the classes, the exercises were designed based on a completion strategy, where students had to complete the program based on a provided template that included definitions of the required variables. After each class, we evaluated the robustness and usefulness of the visualizations by the extended TEDViT using questionnaire surveys administered to the participants and objective verifications by the teaching staff.

From the results of the questionnaire surveys, we found that many participants accepted the visualizations by the extended TEDViT favorably and found them helpful when working on the exercises. From the verifications by the teaching staff, we found that the extended TEDViT could appropriately visualize the behaviors of learners' programs in general after some bug fixes to the system and the drawing rules. These results suggest that the extended TEDViT could visualize the behaviors of learners' own programs with a certain degree of robustness. Therefore, we can conclude that our approach, which reflected teachers' intents of instruction regarding visualizations of learners' programs by using concrete and superficial statuses of the programs' execution, would have a certain degree of validity. We expect to achieve further improvements to the robustness of our approach by continuing the classroom sessions utilizing the extended TEDViT.

## Acknowledgements

This study was supported by JSPS KAKENHI Grant Numbers JP16K01084, JP18K11567.

## 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 (ICCE2014)*, 343-348.

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.

Van Merrienboer, J. J., & Krammer, H. P. (1987). Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instructional Science, 16*(3), 251-285.

Yamashita, K., Fujioka, R., Kogure, S., Noguchi, Y., Konishi, T., & Itoh, Y. (2016). Practice of algorithm education based on discovery learning using a program visualization system. *Research and Practice in Technology Enhanced Learning (RPTEL), 11*(15), 1-19. doi:10.1186/s41039-016-0041-5.

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 (RPTEL), 12*(17), 1-16. doi:10.1186/s41039-017-0058-4.