GUI-Based System for Effortless Program-Visualization Creation Using Time-Series Information

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

Abstract: In this paper, we describe a system that effortlessly creates program visualization (PV) by incorporating time-series information into a graphical user interface (GUI) system for PV creation. Although several PV systems have been developed, only a few have been introduced or used continuously in actual classes. One of the main obstacles to using PV systems in actual classrooms is the significant amount of time needed to integrate PV systems into actual educational setups. We developed a PV system called TEDViT and introduced it into several practical classes. While programming learning with TEDViT had a noticeable effect, the time required for PV customization was a non-trivial problem. To address this issue, GUI-based WYSIWYG PV editor would be a promising approach. However, many existing systems only support PV drawing. We believe that PVs should be more than mere drawings of data structures. They should be sequences of drawings with a program-execution process. This study has therefore developed a PV-creation support system that considers the continuity of drawings by incorporating time-series information into the GUI. An evaluation experiment was conducted to measure the time required to create PVs using our system. The results suggest that our GUI system noticeably improves the efficiency of PV creation.

Keywords: Programming education, program visualization system, program visualization design, educational authoring tool

1. Introduction

Program visualization (PV) is a widely accepted approach for supporting novice learners who find it difficult to obtain a clear image of program behavior. Thus far, several PV systems have been developed, and many positive learning effects have been reported (Pears et al., 2007). However, few PV systems have been introduced continuously in actual classes. One of the main obstacles to continuous use is the time cost involved; teachers who introduce PV systems into their classrooms must design, integrate, and maintain the PV generated by the system alongside their own lesson plans.

To address this issue, we have developed a PV system, the Teacher's Explaining Design Visualization Tool (TEDViT) and conducted several classroom practice sessions using this system (Kogure et al., 2014). One distinctive feature of TEDViT is the fact that it enables teachers to customize PVs, based on their own instruction plans. Through this feature, teachers can design, integrate, and maintain PVs that reflect their own intentions, achieving positive evaluation results that suggest significant learning effects. Compared to existing PV systems, however, this feature incurs additional costs for PV customization. More work is needed to reduce the cost of using PVs.

While many factors may account for the high cost of PV creation, this paper focuses on PV customization—the most direct way to increase the learning effectiveness of PV systems. Tezuka et al. (2016) proposed a way to reduce the cost of PV creation. They developed a GUI-based system that specified the positions and attributes of drawn objects in TEDViT, making PV creations more intuitive—and not based on the numerical specification of coordinates, as previously required. In evaluation experiments, their system reduced the time needed for PV creation by approximately 40%. It is thus a promising cost-reduction approach. However, their system only supports PV drawing. We believe that PV needs to include a sequence of drawings that follow the program execution process, rather than drawing alone.

The present study has developed a system that supports PV creation by incorporating sequences of drawings into the GUI, based on time-series information. To evaluate the effectiveness of this system, we conducted an experiment in which the subjects were asked to create PVs. This paper describes our GUI-based system for effortless PV creation and evaluation experiment. The evaluation results suggest that our approach to PV creation, based on sequences of drawings, is an effective way to support PV creation.

2. Related Works

2.1 Existing PV Systems

During the past few decades, several PV systems have been developed for novice learners, including Python Tutor (Guo, 2013), Jype (Helminen & Malmi, 2010), and PROVIT (Yan, Nakano, Hara, Suga, & HE, 2014). These systems differ in certain ways. For example, Python Tutor runs on a web browser and does not require local installation. Jype provides a learning environment that integrates the PV and automatic-assessment systems for exercise assignments. PROVIT uses 3D graphics in its visualizations. However, all of these systems are similar in the sense that they all visualize the target program and its data structures in a uniform way. Generally speaking, these systems are capable of visualizing programs using a fixed visualization policy. They also allow learners to observe changes in data structure during the execution of each statement. This function is provided by a graphical user interface (GUI), such as next/previous buttons for stepwise execution of the target program. Sorva, Karavirta, and Malmi (2013) provide a comprehensive overview of more than 40 PV systems, which share many similarities.

PV systems demonstrate the runtime behavior of computer programs to novice learners by providing PV that visually encodes data and shows how it is processed in a running program. Novice learners often find it difficult to trace program states and behaviors via data structures. By bridging the gap between their reasoning and computational processes, PV systems can improve novice learners' understanding of programs (Tudoreanu, 2003). However, as Sirkiä and Sorva (2015) have pointed out, PV systems are not always effective. Learners may struggle to understand the meaning of visual elements or neglect important aspects and focus on peripheral elements. We would argue that this reflects a failure to integrate with other materials or offer customizable systems. It also suggests a poor fit with teachers' personal pedagogical styles. Sorva, Karavirta, and Malmi (2013) call this the "problem of dissemination." Teachers' in-class explanations shape the reasoning of learners. Similarly, the designs of PV-system developers shape the way that computational processes are visualized. For PV systems to adequately bridge the gap, they must be designed, integrated, and maintained by teachers.

A few systems, such as ANIMAL (Rößling & Freisleben, 2002), can customize PVs. Prior knowledge and preparation are needed to customize PVs. ANIMAL uses the script language, AnimalScript, to define PV—and the cost of learning it is significant. Moreover, PV creation requires a non-trivial quantity of script code. The sample script for a bubble sort algorithm bundled in ANIMAL consists of 170 lines of script code. More efforts are therefore needed to reduce the cost of PV creation.

2.2 Effortless PV Creation

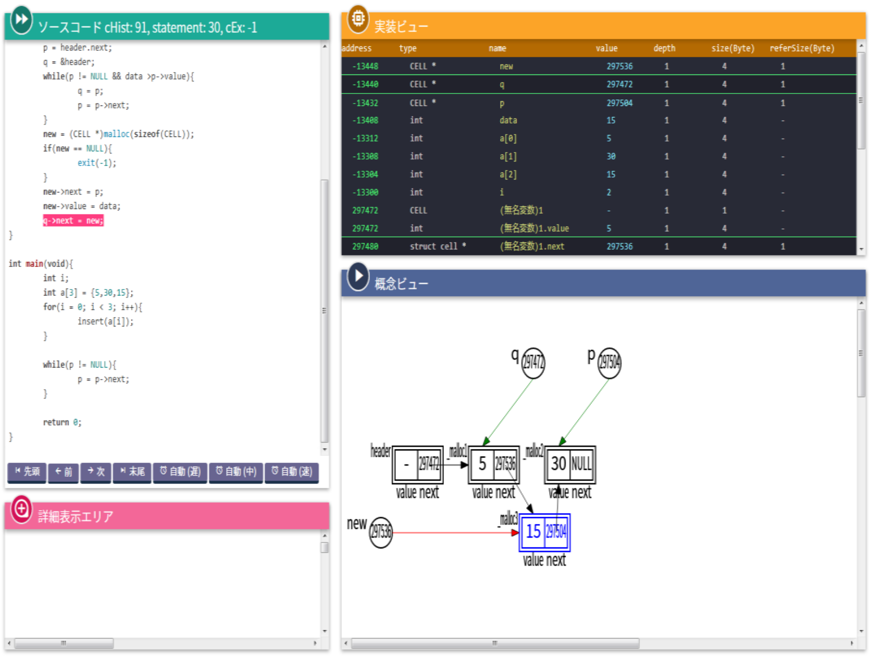
Several studies have investigated ways to reduce the cost of algorithm visualization (AV) and PV creation. Malone, Atkinson, Kosa, and Hadlock (2009) developed a pseudo-code system in which a definition of the visualization can be included in the pseudo code used to represent the target algorithm. The pseudo-code interpreter automatically derives AV from algorithm implementations. This study argues that it is essential to increase AV effectiveness and effortlessness. Velázquez-Iturbide, Pareja-Flores, and Urquiza-Fuentes (2008) developed a system that allows teachers to select PVs from an automatically generated PV sequence in a list format. Although they argue that their system improves effortlessness in PV creation, they do not present any experimental results to prove the point objectively. Rößling and Ackermann (2007) have developed a framework that allows teachers to create AV content on-the-fly by adjusting variable values and attributes. Their framework derives from a set of prepared templates that define the visualization details for various algorithms. These definitions can be saved freely, reducing the effort needed to reuse AVs. PV systems, such as Jeliot 3 (Moreno, Myller, Sutinen, & Ben-Ari, 2004) are often considered effortless because they automatically generate PVs by providing target programs only, although the PVs generated in this way cannot be customized. There are various approaches to effortless PV creation and simple comparisons are difficult to make.

Ihantola, Karavirta, Korhonen, and Nikander (2005) have defined a taxonomy to characterize effortlessness in AV systems. Based on a survey conducted among CS educators, they identify three main categories—*scope, integrability,* and *interaction*—and evaluate several existing systems. The *scope* refers to the range of contexts in which the AV system can be applied—the various algorithmic domains for which the system can be adapted. *Integrability* refers to third-party effortlessness: how easy it is to integrate the AV system into educational setups. *Interaction* is the extent to which the system can be used for different cases. This factor is based not only on interactions between AV content and learners but also on interactions between teachers and content and the extent to which the content is customizable. Although PV systems tend to provide visualizations at a lower level of abstraction than AV systems (Sorva et al., 2013), these three factors also apply to PV creation. PV systems aim to help users understand underlying algorithms by visualizing program behavior.

The present study focuses on the interactions between teachers and content, among other aspects of effortlessness in PV creation, based mainly on using TEDViT in the classroom. TEDViT is a PV system, which allows teachers to customize PV through their own instructions. The practice classes obtained positive learning effects from the interactions between teachers and content (i.e., PV customizations); we can observe students learning to understand programs (Yamashita et al., 2017), extend their class learning style (Yamashita et al., 2016), and so on. The goal of this work is to improve the effortlessness of PV creation by developing a system that supports teachers' PV customizations.

2.3 TEDViT

The TEDViT system interprets each visualization policy by scanning the configuration file and visualizing the PV accordingly. Figure 1 presents a screenshot of the learning environment visualized by TEDViT. The configuration file comprises a set of drawing rules, each of which is a comma-separated value (CSV) entry, consisting of a condition and an object. The condition defines the prerequisites needed to fire the drawing rule. Teachers can use a conditional equation (consisting of a statement ID, variables in the target program, constant values, and comparison operators) to determine the drawing timing. Here, the statement ID is a unique identifier automatically assigned to all statements in the target program by TEDViT. The object defines the operation ("create," "delete," or "update") used to edit the target object and the attributes need to draw it, which include object type, position, color, and corresponding variables. These features make PV customizable in TEDViT by allowing teachers to define the drawing rules. TEDViT supports visualizations of only C programs.



```
        p = header.next;
        q = &header;
        while(p != NULL && data >p->value){
                q = p;
                p = p->next;
        }
        new = (CELL *)malloc(sizeof(CELL));
        if(new == NULL){
                exit(-1);
        }
        new->next = p;
        new->value = data;
        q->next = new;
}

int main(void){
        int i;
        int a[3] = {5,30,15};
        for(i = 0; i < 3; i++){
                insert(a[i]);
        }

        while(p != NULL){
                p = p->next;
        }

        return 0;
}
```

address	type	name	value	depth	size(Byte)	referSize(Byte)
-13448	CELL *	new	297536	1	4	1
-13440	CELL *	q	297472	1	4	1
-13432	CELL *	p	297504	1	4	1
-13408	int	data	15	1	4	-
-13312	int	a[0]	5	1	4	-
-13308	int	a[1]	30	1	4	-
-13304	int	a[2]	15	1	4	-
-13300	int	i	2	1	4	-
297472	CELL	(無名変数)1	-	1	1	-
297472	int	(無名変数)1.value	5	1	4	-
297480	struct cell *	(無名変数)1.next	297536	1	4	1

Figure 1. Screenshot of a Learning Environment generated by TEDViT.

For example, the drawing rule shown in Figure 2 means that when the statement with ID “10” in the target program is executed, TEDViT draws a circle object and assigns it the object ID “OBJ1.” The corresponding variable is “*i*”; hence, the value of *i* is drawn inside OBJ1. OBJ1 is placed in position (x1, y1) with black, white, and black as the line, background, and inner-character colors, respectively, in accordance with the values indicated in the rule. TEDViT provides buttons for stepwise control of the target-program execution, similar to the GUI in typical PV systems. When a learner clicks on the “previous” and “next” buttons, TEDViT finds the corresponding program-execution status, fires the rule for which the condition is satisfied, and visualizes the corresponding drawn objects.

```
state==10, create, OBJ1, circle, i, x1, y1, black, white, black
```

Figure 2. Example of a TEDViT drawing rule.

In classroom practice with TEDViT, the use of PVs that reflected the teacher's instructional intent led to certain positive learning effects. However, the customizability of PV in TEDViT also creates a burden for teachers, who must define the drawing rules. According to Yamashita et al. (2016), it took approximately 30 minutes to define the drawing rules for a single sample selection-sort code, consisting of approximately 30 statements. Although actual teachers rated this as an acceptable class-preparation cost, we consider it a significant burden.

2.4 GUI System for PV Creation

To support the interaction between teachers and content, some systems have functions that design PVs via GUI. Based on WYSIWYG AV editor implementation, Karavirta, Korhonen, Nikander, and Tenhunen (2002) evaluated the effortlessness of existing AV systems. Using TEDViT, Tezuka et al. (2016) developed a GUI system that visually defined the positions and attributes of drawn objects in order to reduce the cost of defining drawing rules. Hereafter, this paper will refer to their system as Tezuka GUI; Figure 3 presents a screenshot of this system.

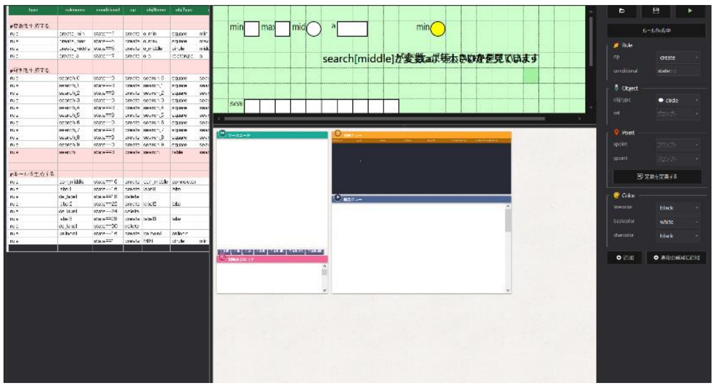


Figure 3. Screenshot of Tezuka GUI.

Tezuka GUI has functions that visually specify the positions of drawn objects, list their available attributes (line color, background color, character color, etc.), specify the values in combo-box style, and highlight grammatical errors in the drawing rules. Tezuka et al. (2016) evaluated the extent to which Tezuka GUI improved effortlessness (measured using the time needed to create drawing rules for TEDViT) and found that the measured times for rule creation were approximately 40% less with Tezuka GUI than without.

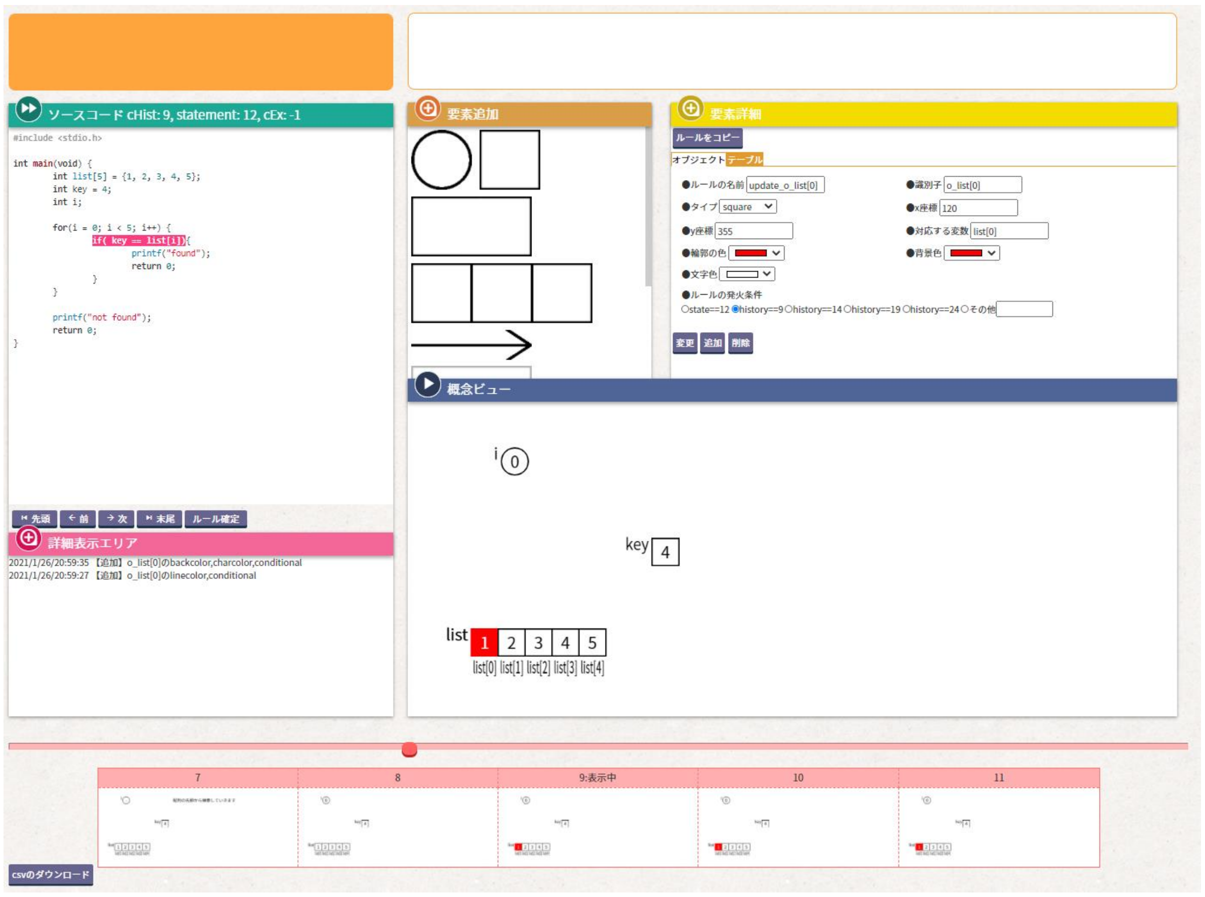
However, Tezuka GUI and existing WYSIWYG PV editors only support PV drawing. In general, PV systems change the drawing content along with the program-execution process. This allows learners to understand the function of each statement in the target program by observing the differences between PVs. Hence, we regard PVs as visualizations of the target domain world. The meaning of each statement in the program is defined by the extent to which executing the statement changes the target domain world. Importantly, PVs are not simply drawings of data structures but sequences of drawings linked to program-execution processes.

PV dynamics are a direct representation of computer-program dynamics, which reveal the trajectory of changes in a computer's internal state, such as data structures changed continuously. By showing these changes directly, dynamic visualizations can offload a learner's cognitive working memory, potentially enabling deeper cognitive processes. Dynamic visualizations can also facilitate cognitive processes that would otherwise require a lot of effort (Kühl, Scheiter, Gerjets, & Edelmann, 2011; Schnotz & Rasch, 2005). In other words, the learning effect of PV systems can be attributed to their dynamism. However, PV editors only support the creation of static visualizations. This study argues that PV creation cannot be fully supported through PV drawings alone. Instead, a function that helps capture a time-series sequence of PVs is required.

Based on this consideration, we aim to support PV creation more effectively by developing a GUI system that includes PV time-series information.

3. GUI System for PV Creation with Time Series Information

Figure 4 presents a screenshot of our developed GUI system, implemented in JavaScript. The *integrability* described in Section 2.1 includes an easy-installation feature.



```
#include <stdio.h>

int main(void) {
        int list[5] = {1, 2, 3, 4, 5};
        int key = 4;
        int i;

        for(i = 0; i < 5; i++) {
                if( key == list[i]){
                        printf("found");
                        return 0;
                }
        }

        printf("not found");
        return 0;
}
```

●ルールの名前 update_o_list[0]
●識別子 o_list[0]
●タイプ square
●x座標 120
●y座標 355
●対応する変数 list[0]
●輪郭の色 🟥
●背景色 🟥
●文字色 ⬜
●ルールの発火条件
○state==12 [✓ ●history==9] ○history==14 ○history==19 ○history==24 ○その他

```
2021/1/26/20:59:35 【追加】o_list[0]のbackcolor,charcolor,conditional
2021/1/26/20:59:27 【追加】o_list[0]のlinecolor,conditional
```

Figure 4. Screenshot of our GUI System.

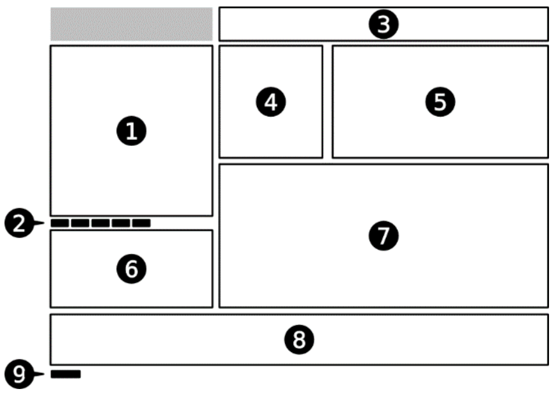


Figure 5. Nine Display Areas in our GUI System.

Our GUI system consists of nine display areas, as shown in Figure 5. The contents of each display area are as follows.

1. The program-code area displays the source code of the target C program to be visualized.
2. The control-button area enables PV creators to have stepwise control of target-program execution.
3. The message area displays operation messages to guide PV creators, for example, by notifying them of insufficient drawing-rule definitions.
4. The drawing-object area is used by PV creators to select a drawn object to be visualized on the PV.
5. The attribute area displays a list of attributes of the selected drawn object, allowing PV creators to set the attribute values using pull-down menus, combo boxes, and input forms.
6. The editing log area displays the editing history of the drawn object and its attribute values.

7. The PV area visualizes the PV by interpreting current drawing-rule definitions.
8. The seek bar and PV thumbnail area displays the current PV and those before and after it in thumbnail style. The PV creator can change the current PV by dragging the seek bar.
9. The save button area enables PV creators to store the PV sequence to a configuration file as a set of drawing rules.

The seek bar and PV thumbnails in Area 8 mainly represent the time-series information described in the previous section. PV creators can arbitrarily change the current PV within the time-series PV sequence by dragging the seek bar. PV thumbnails can make creators aware of the continuity of PVs. They also visually confirm the differences between adjacent PVs in the time series. The seek bar not only helps PV creators navigate the execution process but also helps them grasp the approximate position of the current PV in the time series. We intend to improve PV-generation efficiency using this feature.

4. Evaluation

To evaluate the effectiveness of PV creation using our GUI system, we conducted an experiment to measure the actual time needed to create or modify PVs. A survey by Naps et al. (2002) found that more than 90 percent of participants at the ITiCSE 2002 conference of the ACM cited the time required for PV creation as a factor in their reluctance to use animation (i.e., PV sequences). Thus, the evaluation of effortlessness, based on the time required for PV creation, is considered to be valid. The present study has measured the time required for PV creation using Tezuka GUI and evaluated the degree of improvement in our system's effortlessness.

Ten participants were involved in this experiment: two teachers with experience of teaching programming, five students with experience as programming teaching assistants, and three students with the same level of programming experience as the teaching assistants. We prepared two sample programs as the PV creation targets: a linear-search program and a maximum-value derivation program. Each participant received one sample program and was asked either to create a PV or to modify the PV provided. To reduce the order effects, we specified whether the participants would use Tezuka GUI or our system first. Table 1 briefly summarizes the conditions for each participant.

Table 1. *Conditions for Each Participant*

Participant #	Target	Operation	First use	Second use
1	Linear search	Creation	Tezuka GUI	Our system
2	Linear search	Modification	Tezuka GUI	Our system
3	Maximum value	Creation	Tezuka GUI	Our system
4	Maximum value	Modification	Tezuka GUI	Our system
5	Linear search	Creation	Our system	Tezuka GUI
6	Linear search	Modification	Our system	Tezuka GUI
7	Maximum value	Creation	Our system	Tezuka GUI
8	Maximum value	Modification	Our system	Tezuka GUI
9	Linear search	Creation	Tezuka GUI	Our system
10	Maximum value	Creation	Our system	Tezuka GUI

We began this experiment by explaining to the participants (for 45 minutes) the specification of TEDViT drawing rules and how to use the two systems. Next, we gave them a sample program and a sample PV, without disclosing the drawing rules, and asked them to define drawing rules to reproduce the sample PV. Participants assigned to modification received a set of drawing rules for the sample PV, which included some errors. We measured the time each participant took to define the appropriate drawing rules. Subsequently, we conducted a questionnaire survey on the effectiveness of the seek bar and PV thumbnail function, using a five-point grading system. We also conducted brief interviews to ascertain participants' opinion regarding the two systems.

Table 2 presents the experimental results. Regardless of the order in which they used the GUI systems, target programs, and task operations, all participants took less time to complete the task with our system than with Tezuka GUI. The reduction rate, based on the average time spent by all participants, was 41.3%. This suggests that the PV creation with our GUI system significantly improves effortlessness. However, this experiment compared two full systems, without specifically evaluating the effectiveness of time-series information (the main focus of this study). To obtain more precise results, we must develop two systems, one with a seek bar and PV thumbnail functions and the other without, and measure the time required for PV creation in both systems. We plan to conduct such experiments in the future.

Table 2. *Measured Times for PV Creation/Modification*

Participant #	Time with Tezuka GUI (sec)	Time with our system (sec)
1	1136	925
2	421	249
3	1935	989
4	1360	320
5	914	834
6	523	416
7	1355	845
8	520	377
9	2022	1015
10	1383	819

The questionnaire survey on the effectiveness of the seek bar and PV thumbnail function produced an average score of 4.2, suggesting that the participants gave the function a positive rating. In the interview survey, some participants commented that providing PV continuity alongside the time series made it easier to spot errors. Dividing the task times into two groups, based on participant operations, reduced the average times needed for PV creation and modification by 37.9% and 51.8%, respectively. This suggests that our GUI system supports the task of correcting errors more effectively than Tezuka GUI. This finding will be useful in future evaluations of ways to improve effortlessness in PV creation.

5. Conclusion

The system described in this paper attempts to improve effortlessness in PV creation by incorporating time-series information into a GUI PV-creation system.

One of the main obstacles to the continuous introduction of PV systems in actual classrooms is the significant amount of time needed to integrate PV systems into actual educational setups. Our previous study has developed a PV system called TEDViT and introduced it into several practical classes. While programming learning with TEDViT has a certain effect, the time required for PV customization is a non-trivial problem. Using a GUI-based WYSIWYG PV editor is a promising way to address this issue. However, many existing systems support only the drawing of PVs, even though PVs are not simply drawings of data structures but sequences of drawings alongside a program-execution process. This study has therefore developed a PV-creation support system that takes into consideration the continuity of drawings by incorporating time-series information into the GUI. We conducted an evaluation experiment to measure the time needed to create PVs using Tezuka GUI and our system. The results showed that our GUI system reduced the average time by 41.3%, when compared to Tezuka GUI. This suggests that our GUI system improves the efficiency of PV creation.

The experiment in this study evaluated the effortlessness of the entire GUI system. In future research, we will conduct experiments that focus specifically on the effectiveness of time-series information. We plan to measure the difference between PV-creation times using systems with and without the seek bar and thumbnail function to reflect time-series information. As the participants in the present experiment subjectively felt that the seek bar and thumbnail function had a positive effect on PV creation, we expect to obtain positive results. The results also reveal that our system is more effective than Tezuka GUI in correcting PVs that contain errors. This will be a useful finding as we consider future effortlessness improvements.

Acknowledgements

This study was supported by JSPS KAKENHI Grant Numbers JP18K11566, JP18K11567, and JP19K12259.

References

Guo, P. J. (2013). Online Python tutor: Embeddable web-based program visualization for CS education. *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE)*, 579–584.

Helminen, J., & Malmi, L. (2010). Jype—A program visualization and programming exercise tool for Python. *Proceedings of the 5th International Symposium on Software Visualization*, 153–162.

Ihantola, P., Karavirta, V., Korhonen, A., & Nikander, J. (2005). Taxonomy of effortless creation of algorithm visualizations. *Proceedings of the First International Workshop on Computing Education Research*, 123–133.

Karavirta, V., Korhonen, A., Nikander, J., & Tenhunen, P. (2002). Effortless creation of algorithm visualization. *Proceedings of the Second Annual Finnish/Baltic Sea Conference on Computer Science Education*, 52–56.

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.

Kühl, T., Scheiter, K., Gerjets, P., & Edelmann, J. (2011). The influence of text modality on learning with static and dynamic visualizations. *Computers in Human Behavior*, *27*(1), 29–35.

Malone, B., Atkinson, T., Kosa, M., & Hadlock, F. (2009). Pedagogically effective effortless algorithm visualization with a PCIL. *Proceedings of the 39th IEEE International Conference on Frontiers in Education Conference*, 1501–1506.

Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot3. *Proceedings of the Working Conference on Advanced Visual Interfaces*, 373–376.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velázquez-Iturbide, J. (2002). Exploring the role of visualization and engagement in computer science education. *Working Group Reports from the 2002 Conference on Innovation and Technology in Computer Science Education*, 131–152.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devin, M., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, *39*(4), 204–223.

Rößling, G., & Ackermann, T. (2007). A framework for generating AV content on-the-fly. *Electronic Notes in Theoretical Computer Science*, *178*, 23–31.

Rößling, G., & Freisleben, B. (2002). ANIMAL: A system for supporting multiple roles in algorithm Animation. *Journal of Visual Languages & Computing*, *13*(3), 341–354.

Schnotz, W., & Rasch, T. (2005). Enabling, Facilitating, and Inhibiting Effects of Animations in Multimedia learning: Why reduction of cognitive load can have negative results on learning. *Educational Technology Research and Development*, *53*(3), 47–58.

Sirkiä, T., & Sorva, J. (2015). Tailoring animations of example programs. *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, 147–151.

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), 1–64.

Tezuka, D., Kogure, S., Noguchi, Y., Yamashita, K., Konishi, T., & Itoh, Y. (2016). GUI based environment to support writing and debugging rules for a program visualization tool. *Proceedings of the 24th International Conference on Computers in Education (ICCE2016)*, 303–305.

Tudoreanu, M. E. (2003). Designing effective program visualization tools for reducing user's cognitive effort. *Proceedings of the 2003 ACM Symposium on Software Visualization*, 105–114.

Velázquez-Iturbide, J. Á., Pareja-Flores, C., & Urquiza-Fuentes, J. (2008). An approach to effortless construction of program animations. *Computers & Education*, *50*(1), 179–192.

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

Yan, Y., Nakano, H., Hara, K., Suga, S., & HE, A. (2014). A C programming learning support system and its subjective assessment. *Proceedings of 2014 IEEE International Conference on Computer and Information Technology*, 561–566.