Research and Practice in Technology Enhanced Learning
Vol. 6, No. 2 (2011) 83 – 106

DEVELOPMENT AND EVALUATION OF A NEW PRESENTATION SOFTWARE PROGRAM (CODEX) FOR TEACHING PROGRAMMING CODE

HIDEKAZU KAMINISHI

MASAO MUROTA

Graduate School of Decision Science and Technology, Tokyo Institute of Technology,
Ookayama 2-12-1, Meguro-ku, Tokyo, 152-8552, Japan
{hideka,murota}@mr.hum.titech.ac.jp
http:/www.mr.hum.titech.ac.jp

In this study, we developed a presentation software program (CodEx) for web programming language courses. It has functions to display sample source code, edit it, and display its execution result all in one slide. To evaluate the efficiency of this software (CodEx), two experiments were conducted. First, reviews done by nine teachers proved its efficiency in teaching programming language. Secondly, to evaluate the effectiveness of CodEx, we conducted a micro teaching experiment which compared our proposed teaching method (CodEx) with the conventional method, based on tests and a questionnaire. The tests included fill-in-the-blank questions, and tests of reading and writing programming codes. In the reading test, students taught with CodEx obtained higher scores than those taught by the conventional method. Our findings suggest that this software is better at helping students to understand the mechanism of programming codes.

Keywords: Presentation software; Programming language education; Split-attention effect

1. Introduction

1.1. *Background*

1.1.1. *Education and presentation software*

Presentation software and devices are commonly used. The most popular presentation software program is Microsoft PowerPoint. Microsoft estimated that about 1.25 million PowerPoint presentations were given every hour in 2004 (Levasseur & Sawyer, 2006). Generally a presentation has many constraints, the most relevant being the limited time to present consistent information (Savoy, 2009).

Although there are no compelling results to prove or disprove that PowerPoint is more effective for learner retention than traditional manual presentation methods, various educational presentation tools which solve some weaknesses or problems of PowerPoint have been developed and used recently. “Classroom Presenter”, a tablet PC-based presentation system (Anderson *et al.*, 2004), is one such example. This system combines the advantages of existing computer-based systems with those of traditional manual

presentation systems. The program introduced functions of common practical use for lectures, such as instructor's notes, interactive writing, diagrammatic use of ink, and attention-getting techniques. "Fly" (Holman *et al.*, 2006) is a presentation system with Mind Maps which introduces spatial organization to presentations. This software enables one to show hierarchical structures within a presentation. This method was created as a solution to the critique of Tufte (2003) which discussed the problems of using PowerPoint. "Pallette" (Nelson *et al.*, 1999) is a presentation system with index cards that are printed with slide content that is easily identified by both humans and computers. A presenter composes a talk by selecting cards from one or more sets. This system supports interactivity between teacher and students.

A lot of presentation software is developed and used in education. This is also common in source code education. In a class on computer programming language, the computer is an indispensable device for teaching. As Buck and Stucki (2001) and Kölling and Rosenberg (2001) insisted, the teacher should give beginning students some examples of programming codes. One popular teaching method is for the teacher to present sample code using presentation tools and have the students work on coding that imitates the sample code. Teachers commonly use commercial or free programming environment software programs such as Dreamweaver or Aptana to show the students' executed results. The problem with this method is that students often do not understand the sample code and thus are not able to produce the expected results. For this reason, teachers should display the execution results of the sample codes they have presented, preferably on the same screen or slide as the sample code.

There is a common problem when teachers display execution results; teachers have to switch among different applications: presentation software to deliver the lecture, a text editor to show sample codes, and a programming environment to show the corresponding results of execution. This might cause a split-attention effect in learners as described in the following section. Therefore, this study deals with two specific purposes related to the instructor's problem of needing to use multiple applications at the same time: first, the usability of the application by teachers; second, the efficiency of the application for students. In section 1.2 these objectives are described accordingly.

1.1.2. *Teaching computer programming and the split-attention effect*

Split-attention occurs in the classroom when learners are required to mentally integrate several sources of physically or temporally disparate information and thus split their attention among the sources. Learners studying information that is presented from multiple sources (i.e. screens, slides, or programs) in the same lesson are outperformed by learners studying the same information presented in an integrated format. When information is not integrated, cognitive load is increased by the need to mentally integrate multiple sources of information. This increase in extraneous cognitive load is likely to have a negative impact on learning compared to conditions where the information has been restructured to eliminate the need to split attention. When presenting disparate sources of information that must be mentally integrated in order for the information to be

understood, those sources of information should be presented in an integrated format (Ayres & Sweller, 2005). Several studies of split-attention effect in multimedia learning have been conducted. A work of Tarmizi and Sweller (1988) is an example in geometry problem acquisition. The researchers scaled and compared the students' elapsed time for acquisition and answer, and the post-test score when they provided materials using an integrated format and when they did not. They showed that an integrated format reduces the time needed to acquire new skills and concluded that an integrated format reduces the cognitive load. This research examined the difference between the split-attention effect found when studying visual information and textual information which have different properties of information.

The works of Sweller and Chandler (1994) and Chandler and Sweller (1996) provide examples in computer-based learning. They concluded that paper-based fully integrated instructions for a computer application were superior to simultaneous use of a computer. These results suggest that learning with a computer is not always the most effective. They remarked that this was because of the increase in cognitive load caused by split-attention. Following the success of the integrated manual approach, Cerpa *et al.* (1996) reasoned that its effectiveness could be plausibly caused by differences in media rather than split-attention. They found a split-attention effect with integrated materials and conventional ones using spreadsheet learning. They compared integrated computer-based educational software with conventional manuals used with computer software. They showed that integrated software reduced the split-attention effect more than conventional programs. In the post-test, the integrated group (computer program only) significantly outperformed the conventional groups in the test questions tapping knowledge that was high in element interactivity, such as creating a formula. This study also demonstrated that computers could be used effectively provided the split-attention effect was avoided. These studies compared the split-attention effect among different media.

Multiple sources of information that must be integrated to be understood can also be separated in time, resulting in temporal separation. It is reasonable to suppose that temporal separation also generates an extraneous cognitive load for exactly the same reasons as physical separation (Ayres & Sweller, 2005). For example, Mayer and Sims (1994) found evidence that a concurrent presentation and animation was superior to a sequential presentation of either narration followed by animation or animation followed by narration on problem-solving tasks. Mayer (2001) argued that because of the limitations of working memory, words and pictures should not be separated temporally.

Our purpose is to reduce the split-attention effect in presentations of lectures on computer programming. As mentioned before, presentation software for delivering the lecture, a text editor for showing sample codes, and a programming environment for showing the results of execution are all essential elements of such lectures. The problem is that it is difficult to present these applications all at the same time, which means that teachers present them in a temporally separated manner. One solution to reduce the split-attention effect is to use dual screen environments. There are some studies on presentation software that uses a dual screen environment.

1.1.3. *Computer programming lectures with multi screen*

For example, "MultiPresenter" (Lanir *et al.*, 2008) is a presentation system designed to work on very large display spaces (with multiple displays or physically large high-resolution displays). This system enables teachers to show various slides simultaneously. However, this system is not concerned with showing the computer programming environment. Chang *et al.* (2009) tried to reduce the temporal split-attention effect by using a dual screen to display presentation software and the programming environment at the same time. They divided their examinees into two experimental groups. One group was taught in a traditional classroom with a single screen, while the other group was taught with a dual-screen setup. They showed that the dual-screen group scored significantly higher based on the results of a pre-test and a post-test.

Endo and Murota (2010) developed a dual-screen presentation tool with zooming user interface (ZUI). This tool allows teachers to show several application windows, such as presentation slides and the programming environment on a dual screen. However, this method requires a dual-screen setup which is often not available for the classroom. In addition, this method may cause a spatial split-attention effect. A single screen environment seems to be preferable for students when possible. Therefore, we have developed a single-screen presentation software program which allows teachers to simultaneously display both sample code and its corresponding execution results on a single slide.

1.2. *Purpose of this study*

In this study, we developed a presentation software program with the following features:

Facilitating students' learning by reducing the split-attention effect: We aimed to reduce the split-attention effect on students and facilitate their learning by displaying both a source code and its execution results on the same screen simultaneously.

Reducing the need to switch between applications: With our presentation software, teachers can display both the sample code itself and its execution results in the same application. This function enables teachers to display both code and results simultaneously without worrying about switching back and forth between applications.

2. Feature and Function of the Software CodEx

2.1. *Features of CodEx*

Our new presentation software program, CodEx, consists of an environment that makes it possible to both edit and execute programming code in one slide. The code and the results of executing that code are displayed in the same screen at the same time. CodEx stands for Code and Execution displayed simultaneously in one single slide.

CodEx is an HTML presentation software program, based on W3C Slidy (Raggett, 2005). We have expanded the functions of that software.

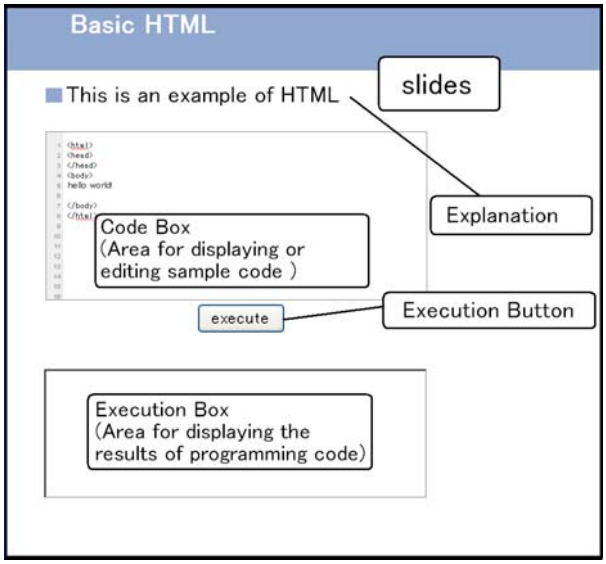

Figure 1. Layout of a slide and its component in CodEx.

To create presentation slides with CodEx, the user edits a text file with Wiki-like notation. This notation was previously proposed by Kaminishi and Murota (2009), but we have expanded it with new functions. The user can convert Wiki-like notation into an HTML presentation file by using converter script.

An example of a possible slide layout in CodEx is shown in Figure 1. The slide in Figure 1 has a text of a sample slide, a "code box", an "execution box", and an execution button. The "code box" is the area where sample code is displayed and edited. The "execution box" is the area where the execution results for that code are displayed.

To display the execution results of the sample code in the "execution box", the user simply clicks the "execution button". It is possible to edit this sample code without switching to another application. It is always possible, with just one click, to display in the execution box the result of any changes made to the sample code.

2.2. *Components and functions of CodEx*

2.2.1. *Components on a slide*

Code box: The "code box", as mentioned above, is the area for displaying and editing sample code. HTML/CSS/JavaScript/PHP (requires web server for execution) is now available.

Execution box: The "execution box" is the area which displays the execution results of the code in the code box. The execution box is made up of the "iframe", one of the HTML elements. In other words, HTML for showing results (HSR) is displayed there.

Executable code is executed and displayed in HSR. HSR must import the JavaScript library named "exectext.js" to execute the code from the code box. This file includes functions which enable the user to execute code such as the manipulation of DOM (Document Object Model), and so on. A teacher can similarly prepare HTML sample code as usual except he or she must add one line of code to import the statement of library.

Teachers who use CodEx have to prepare the HSR files before making the slides. Generally, one can display HTML pages out of domain in iframe. However, we must set local HTML file in the execution box, because of the limitation caused by the web browser's security restrictions.

The function of executing code: The contents (executable text code) written in the code box are transferred to HSR and displayed in the execution box. HSR reads exectext.js as mentioned before. Functions in exectext.js drive these codes that are executed by set language (HTML/CSS/JavaScript), then the results are displayed in the execution box.

Execution button: The execution button (labeled "execute") is displayed below the code box. When execution button is pressed, the executable code in the code box is executed, and the result is displayed in the targeted execution box. This "execute code function" enables teachers to edit sample codes, and execute them in a slide.

2.2.2. *Execution according to each language*

In this section, we explain the functionality of each mode of computer programming language as used in CodEx. The word "execution (execute)" is used to mean "displaying HTML" or "applying CSS" in this paper.

HTML mode: In HTML mode, CodEx displays the execution results of HTML code in the execution box. When one particular portion of HTML code has already been displayed in the code box, it will be replaced by the new result of execution, it will not be appended.

HSR (HTML for showing results) reads the JavaScript library execute.js. HSR receives executable codes in the code box; then, it displays the result with the scripted functions.

As an executable code of HTML, code with <body> tag or without <body> tag (partial HTML) is either acceptable. Teachers can show each type of HTML. The latter case enables teachers to display only the necessary portion of code. The functions scripted in exectext.js recognize the two types of HTML code. When executing HTML code, the functions in exectext.js analyze the code. They distinguish whether the code includes <head>, <body> tag or not.

If the received HTML code does *not* have <body> tags, the functions operate the DOM of HSR. That is to say, they replace the <body> tag's "innerHTML" property of HSR, with the received code.

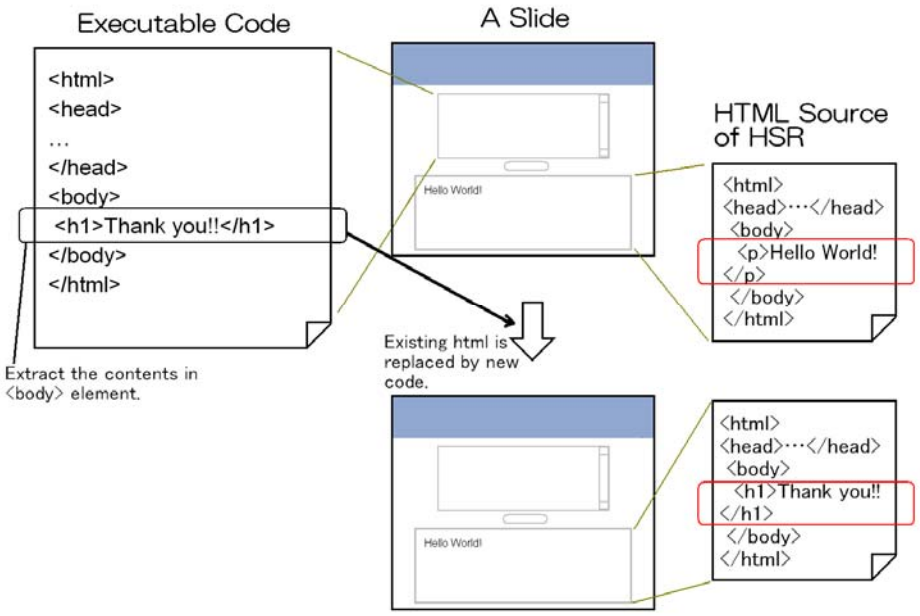

Figure 2. Displaying HTML code with <body> tag in execution box.

If the received HTML code has a <body> tag, the portion of the inner <body> tag is extracted. Then, the resulting process is similar to the latter case. The features of this process are shown in Figure 2.

In this case, the contents in <head> tag are extracted and reflected in HSR. If a <title> tag is included, the functions replace the existing <title> tag with a new one. If a <style> tag is included, the functions pass the style data to the function for applying CSS, which is described in the follow section.

CSS mode: In CSS mode, we can see the applied results of CSS in HSR displayed in the execution box. If the user presses the execute button, he or she can change the style of HSR. In this mode, existing CSS script is replaced by a new code, as in HTML mode (not appended, but overwritten).

HSR displayed in the execution box (CSS is applicable too) is available in two types: a prepared HTML file before making slides, and a displayed result through the function of HTML mode. In the latter case, we must show the two code boxes (for HTML and CSS) and the execution box all on one slide. In this case, one execution box is targeted by two code boxes as shown in Figure 3.

Additionally, if the user scripts CSS as embedded style in HTML, he or she must choose HTML mode instead of CSS mode.

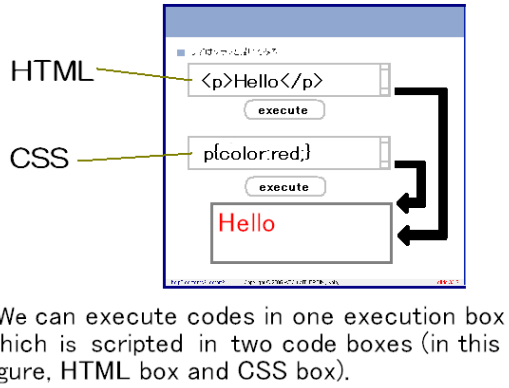

Figure 3. An example of execution of HTML and CSS in one execution box.

The process driven by the function in exectext.js is a replacement of existing CSS data with new data in executable code. The function deletes existing <link> tags or <style> tags in HSR related to CSS, and then appends new CSS code received from the code box.

JavaScript mode: In JavaScript mode, we can see the results of what is displayed by the execution of JavaScript code in the code box. The user may work with the functions of HTML or CSS. In this case, additional code boxes are required. Figure 4 shows the example of executing in JavaScript mode. In this figure, an alert message is shown; this alert occurs in HSR, not in HTML.

In JavaScript mode, HSR receives code from the code box. Then, the functions process the code, executing it as JavaScript and the results are displayed in the execution box. The functions pass the code as an argument to eval( ) function (equipped function in JavaScript), which evaluates the string data as JavaScript code.

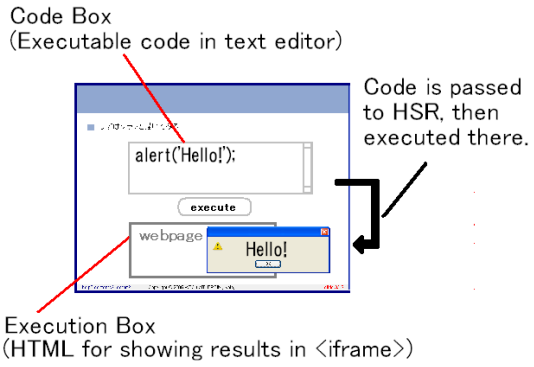

Figure 4. Execution of JavaScript.

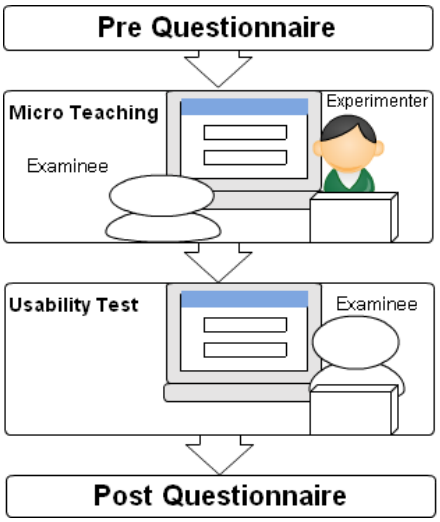

Figure 5. Procedure of Experiment 1.

3. Experiment 1: Teachers Review of CodEx Software

3.1. *Procedure of the experiment*

To prove the efficiency of CodEx for teachers, we conducted experiments for review. The procedure was as follows:

The subjects were nine people who have experience in teaching computer classes in universities. The experiment included a micro teaching session and a usability test, with pre/post questionnaire. The procedure is illustrated in Figure 5.

Before starting the experiments, an experimenter explained to the subjects how to use CodEx, and let them use it for several minutes. After that, the subjects were asked to fill in a pre-questionnaire.

In a micro teaching session, the experimenter taught a section of a programming course to the subjects. The content of the class was “Introduction to CSS”, which included the grammar of CSS with examples. The experimenter then showed another related example (for example, the experimenter replaced the class selector of CSS with another one in a sample code) using the function for editing codes.

The course target was university students whose major was not computer science. We chose the content because sample source codes are collaborated with HTML and CSS. Subjects were provided with printed paper handouts of the slides and a pen. They could use a computer with CodEx software that had the same slides as the experimenter did. Or, they could watch the slides on the computer and try to use CodEx software to, for example, type source code and display its execution results.

In a usability test, the subjects were asked to recreate the experimenter's operations from the micro teaching session with CodEx software. Afterwards, the subjects were asked to complete a post-questionnaire.

3.2. *Results of questionnaire*

For the pre-questionnaire, all subjects used conventional presentation tools to show slides and application software to give a lecture. They had to switch between the presentation software and application software to display executed code. The score of the question on whether or not changing applications caused interruption for the lecture was 3.67 (SD: 1.05), which is a wide deviation.

Results of the post-questionnaire (maximum score five points) are shown in Figure 6. Figure 6 shows the result of the test of hypothesis on the mean (null hypothesis: the average of the questionnaire is equal to three points).

Questions 1 and 4, related to visual issues, scored low, because of the small size of characters.

Questions 6 to 10, related to the main purposes of this study, scored significantly higher than three points in $p < 0.01$ level. They are appreciable results in the evaluation.

The teachers made the following suggestions in the "Comments" section of the questionnaire: display slides without scrolling (five teachers); show error messages (three teachers); support other languages (two teachers).

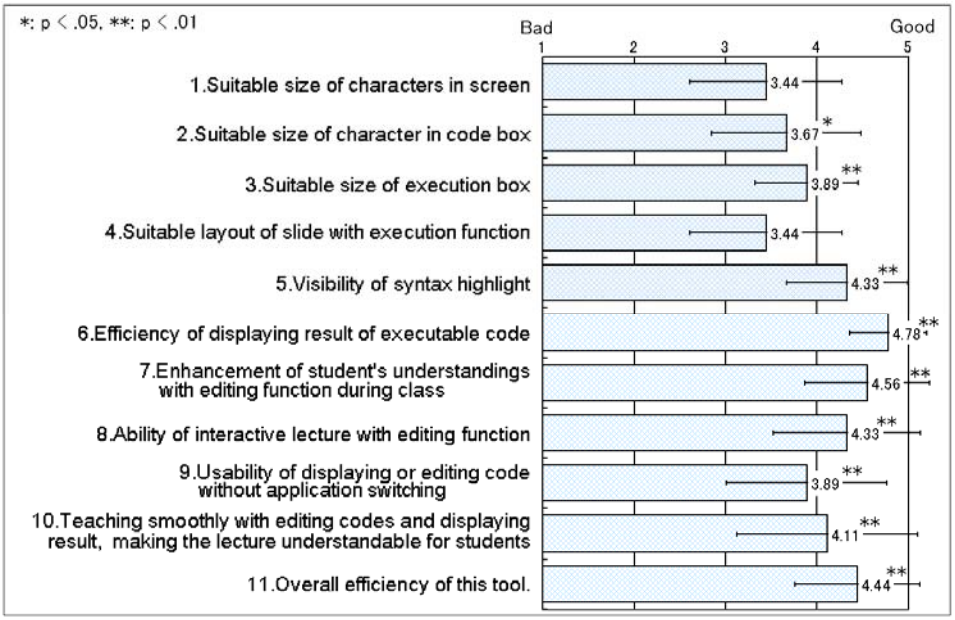

Figure 6. The results of post-questionnaire (1:bad; 5: good).

3.3. *Considerations*

3.3.1. *Displaying slides*

In any programming lecture, characters on the screen tend to appear small. This experiment was not an exception, the teachers did not give high scores to CodEx when asked about visual issues.

When implementing new functions for the screen (in this case the execution box), the area left remaining to display conventional content (in this case the sample code) inevitably becomes small. To solve this problem, we should use a larger screen with higher resolution, or use two or more screens.

3.3.2. *Usability of displaying and editing code without application switching*

Question 9 asked the teachers to rate the "usability of displaying or editing code without switching between applications". Teachers gave CodEx high scores in this area. CodEx was beneficial to the teachers in giving their lesson as compared to conventional software.

3.3.3. *"Comments" section of the questionnaire*

The problem of scrolling was mentioned as stated above.

The issue of displaying error messages may be solved by a debug tool. Firebug, which is an add-on of Firefox, is a tool for notification of errors in HTML/CSS/JavaScript, *et cetera* (Hewitt, 2011). But at present, some messages in the CodEx program still appear unclear, consequently some modifications are required.

To support other languages, we may use "ideone API". This is an online compiler and debugging tool which allows the user to compile and run code online in more than 40 programming languages (Sphere Research Labs, 2010). The results can be displayed in a web page.

3.4. *Modifications of the software*

Per the teachers' suggestions, we have modified CodEx as to visual issues. Specifically, the default font size of the presentation tools was set larger. The margins in each element were reduced. Furthermore, the complicated layout for slides that include a figure, a code box, an execution box and explanatory text in the same slide are now available with converter. This modification reduces the need to scroll through the slides.

4. Experiment 2: Investigation of the Impact on Students

After tackling the display issues pointed out by the teachers in the above experiment, we conducted another experiment to evaluate CodEx software from the point of view of students.

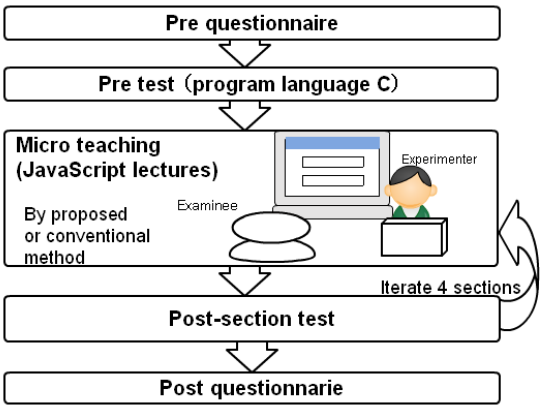

Figure 7. Procedure of Experiment 2.

4.1. *Purpose of the experiment*

To evaluate the impact of using this software, we conducted an experiment which compared the effectiveness of a lesson taught with CodEx (the proposal group) and without CodEx (the conventional group). We used micro teaching lessons to address this question. One of the authors of this paper taught a lecture, “Object on JavaScript”, to subjects and evaluated them with a pre-test and post-test, and a pre-questionnaire and post-questionnaire.

4.2. *Procedure*

Figure 7 shows a flowchart of the procedure of this experiment. The participants were divided into two groups randomly. The experiment included a pre-questionnaire before the pre-test. So that the subjects would only be required to commit to one day of participation, we conducted the pre-questionnaire and the pre-test on the same day as the post-test and post-questionnaire. The experiment was designed to analyze the effect on participants who had no experience with JavaScript. Therefore, those participants who did have experience with JavaScript were not considered in our analysis. However, we let all participants complete the entire experiment in order to avoid any interruption that could affect the test-takers during the course of the experiment. At the end of the experiment, we only took into account the data of those participants who did not have experience in JavaScript. As a result, the number of participants per group turned out to be different: eleven participants in group one, and five participants in group two.

In detail, all participants answered a pre-questionnaire, then they took a pre-test about C programming language (C). Next, participants attended micro teaching lectures, which included four sections. These lectures were taught via the proposed method (with CodEx software) or conventional one (with conventional presentation tools, text editor, and browser to show the results of JavaScript code). The theme of each lecture section and the teaching methods used are shown in Table 1.

Table 1. Teaching methods for groups in each section.

Theme of section	Group 1 (n = 11)	Group 2 (n = 5)
1. Variables and Functions on JavaScript	Proposal	Conventional
2. Object, Property and Method on JavaScript	Conventional	Proposal
3. Copy of Variables, Call by Variable/Reference	Proposal	Conventional
4. Constructors on Javascript	Conventional	Proposal

For the proposed method, as shown in Figure 8, the teacher (experimenter) showed sample code in the code box, its explanation, and the execution box together on one screen. Students could see all these features at the same time.

For the conventional method, the teacher (experimenter) had to switch among different applications during the lecture. As shown in Figure 9, the teacher had to switch back and forth among the presentation software, the text editor, and the browser to present each sample code. During the lecture, students could not view these features at the same time.

The lecture contents and the sample codes used were the same for both the proposal group and the conventional group.

The participants took tests after each section of the lecture. These were paper-based tests. The contents of the tests were related to the lecture. After all of the participants finished the test the experimenter started teaching the next section of the lecture.

Examinees took tests after each section. Those were paper tests. Contents of the tests were related to the teaching session. After all of the examinees finished to answer, the experimenter started teaching the next section.

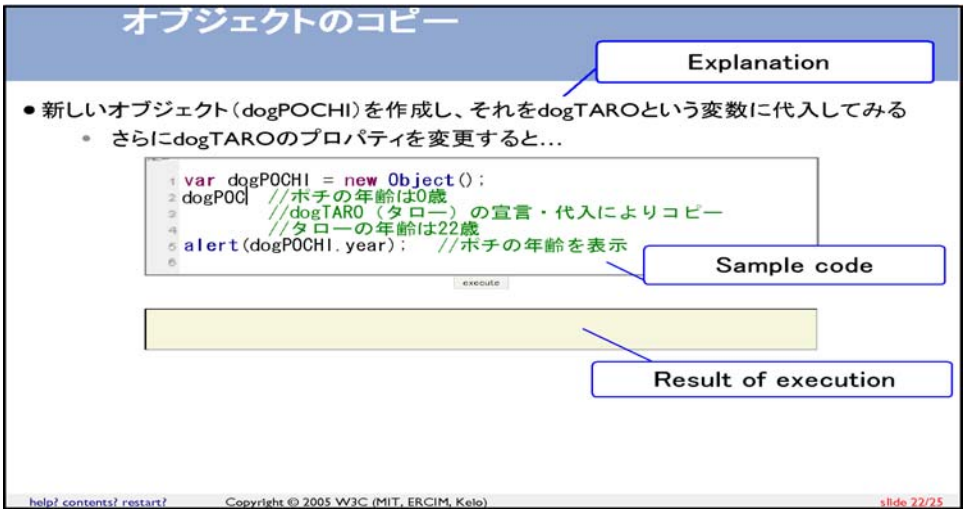

Figure 8. An example of slide in the proposed method.

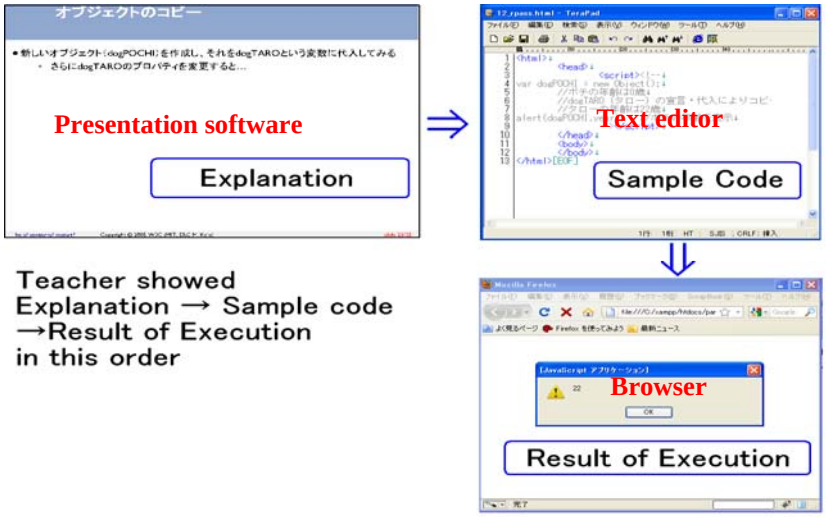

Figure 9. Teaching method with the conventional method (with the need of switching applications).

4.3. *Contents of the pre-test, lecture, post-test and post-questionnaire*

Pre-test: The pre-test asked questions on basic knowledge of C programming language. All participants had some experiences with C. They were asked to show the execution results of a portion of sample code, and also how to use printf statement, treatment of variables, declaration of a function and treatment of pointer variables. These questions were related to the JavaScript lecture.

Contents of the lecture: The topic of the lecture was "Object on JavaScript". The lecture was separated into four sections. The contents of these sections are shown in Table 2. As shown in Figure 10, for both groups the experimenter taught sections 3 and 4, which included complicated explanations of the mechanism of programming execution, while inputting sample code on the screen. To control the experimental conditions for each group, no questions were allowed from the participants. Table 3 shows lecture time (both the assumed time from the guidance plan and the actual time elapsed for each experiment), number of slides, number of sample codes and number of edited codes presented in each section.

Post-section tests: The participants took post-section tests after each section of the lecture. Each test included a verbal test (a fill-in-the-blank test: five points maximum), a reading test (to point out the error in code or provide the correct result of a portion of fixed code: five points maximum), and a writing test (to write JavaScript code: five points maximum). Each test was a 15 point test. The tests were not time limited, but participants were asked to answer rapidly and correctly. Participants were not allowed to re-answer questions which they had already answered before.

Table 2. Content of lecture and post section test.

Section	Contents of lecture	Contents of post-section test (5 points max. each)
1	Declaration and substitution of variables on JavaScript	Filling in blanks
	Declaration of functions in two ways on JavaScript	Checking and correcting given codes Description codes of declaration of functions
2	Ways to object generation, property and method on JavaScript	Filling in blanks Checking and correcting given codes Description codes of declaration of objects
3	Copy of primitive type variables and object variables (explain with pictures)	Filling in blanks Variable values after executed codes Description codes of copy of variables
4	How to use Constructors on JavaScript	Filling in blanks Variable values after executed codes Description codes with constructors

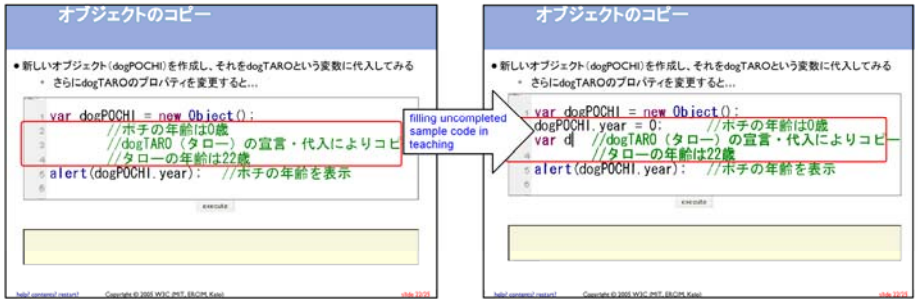

Figure 10. Filling uncompleted sample code.

Table 3. Time of lecture, number of slides and sample codes in each section.

Section	Time of lecture			Number of slides	Number of sample codes	Number of edited sample codes
	Assumption time in guidance plan	Actual time in proposed method	Actual time in conventional method			
1	8 min	7 min 51 s	7 min 3 s	6	6	0
2	8 min	8 min 19 s	8 min 52 s	7	4	0
3	8 min	7 min 41 s	7 min 28 s	4	3	3
4	8 min	7 min 55 s	7 min 31 s	4	3	3

4.4. *Results*

4.4.1. *Experimentees*

Examinees were 21 male undergraduate students with major in computer science in a university in Japan. They had studied C programming language in their curriculum, although no JavaScript lecture had been included in their curriculum. In this experiment, students who on a pre-questionnaire mentioned that they have experience with JavaScript or had not answered the question about their experience with JavaScript were not taken into account in this investigation. Therefore we just considered 16 students.

4.4.2. *Pre-questionnaire and pre-test*

In the pre-questionnaire, we asked participants about their experience with programming. According to their responses, we considered only participants who did not have experience with JavaScript.

The numbers of participants was 11 in group 1 and five in group 2. The average years of programming experience was 2.41 years in group 1 (SD: 1.69) and 2.80 years in group 2 (SD: 2.16). There are no significant differences ($F(12,5) = 0.47$, $p = 0.70$).

In the pre-test (five points maximum), we measured each participant's understanding of C to gauge their base for understanding JavaScript, which is a common assumption of programming instructors. Most of the participants made mistakes on the questions related to pointer variables (which are considered to be related to the reference type value of the JavaScript concept). This problem cannot be solved without comprehensive understanding of pointer. The average score of group 1 was 3.90 (SD: 1.14), while that of group 2 was 4.00 (SD: 0.71). There was no significant difference in score between the groups ($F(11,5) = 0.37$, $p = 0.87$).

4.4.3. *Section 1*

The result of the post-section test of section 1 is shown in the top level of Table 4.

On the fill-in-the-blank test, the proposal group (using CodEx) scored higher than the conventional group (using conventional presentation software), however no statistical significance was found. The effect size (Cohen's d) was small ($0.2 \leq d < 0.5$).

However, on the reading test, the conventional group scored higher. This is likely due to a "ceiling effect", because both groups scored very high (very few participants made mistakes).

On the writing test, the proposal group scored higher than the conventional group, at a $0.05 < p < 0.10$ level. The effect size was large ($0.8 \leq d$). The standard of effect sizes applied in this paper is based on Cohen (1988).

In total, the proposal group scored higher at a $0.05 < p < 0.10$ level. The effect size was large ($0.8 \leq d$).

Table 4. The results of the post-section test.

Section 1								
	Fill-in-blanks		Reading		Writing		Total	
	P	C	P	C	P	C	P	C
Average	4.73	4.60	4.91	5.00	1.73	0.60	11.36	10.20
S.D.	0.65	0.55	0.30	0.00	1.42	0.89	1.29	0.84
p-value by t-test	0.709		0.519		0.128		0.088+	
Effect size	0.21		-0.43		0.95		1.07	
Section 2								
	Fill-in-blanks		Reading		Writing		Total	
	P	C	P	C	P	C	P	C
Average	4.20	4.55	4.80	4.36	2.60	4.27	11.60	13.18
S.D.	1.30	0.69	0.45	0.81	1.95	1.19	1.82	1.47
p-value by t-test	0.492		0.283		0.050+		0.084+	
Effect size	-0.33		0.67		-1.04		-0.96	
Section 3								
	Fill-in-blanks		Reading		Writing		Total	
	P	C	P	C	P	C	P	C
Average	3.36	4.20	4.45	4.20	3.09	4.20	10.91	12.60
S.D.	1.80	0.84	0.69	0.84	1.58	0.45	2.63	1.14
p-value by t-test	0.346		0.530		0.151		0.195	
Effect size	-0.59		0.33		-0.96		-0.84	
Section 4								
	Fill-in-blanks		Reading		Writing		Total	
	P	C	P	C	P	C	P	C
Average	3.20	4.09	3.80	3.36	4.20	3.73	11.18	11.20
S.D.	1.79	1.04	0.84	0.92	1.30	1.27	2.64	3.03
p-value by t-test	0.225		0.450		0.357		0.901	
Effect size	-0.61		0.49		0.25		-0.01	
Total								
	Fill-in-blanks		Reading		Writing		Total	
	P	C	P	C	P	C	P	C
Average	3.94	4.34	4.56	4.13	2.72	3.41	11.22	11.88
S.D.	1.50	0.83	0.67	0.91	1.71	1.72	2.11	2.25
p-value by t-test	0.186		0.032*		0.114		0.233	
Effect size	-0.34		0.55		-0.40		-0.30	

Notes: [a] P: proposal group; C: conventional group
[b] Effect size are Cohen's *d*
[c] *: $p < 0.05$; +: $0.05 < p < 0.10$)

4.4.4. *Section 2*

The result of the post-section test of section 2 is shown in the second level of Table 4.

On the fill-in-the-blank test, the conventional group scored higher than the proposal group, however no statistical significance was found. The effect size was small ($0.2 \leq d < 0.5$).

On the reading test, the proposal group scored higher although there was no statistical significance. The effect size was medium ($0.5 \leq d < 0.8$).

On the writing test, the conventional group scored higher than the proposal group, at a $0.05 < p < 0.10$ level. The effect size was large ($0.8 \leq d$).

In total, the conventional group scored higher at $0.05 < p < 0.10$ level, and the effect size was large ($0.8 \leq d$).

4.4.5. *Section 3*

Table 4 shows the results of the post-section test for section 3.

On the fill-in-the-blank test, the conventional group scored higher than the proposal group, however, no statistical significance was found. The effect size was medium ($0.2 \leq d < 0.5$).

On the reading test, proposal group scored higher although there was no statistical significance. The effect size was small ($0.2 \leq d < 0.5$).

On the writing test, the scores of the conventional group were higher than the scores of the proposal group, however no statistical significance was found. The effect size was large ($0.8 \leq d$).

In total, the proposal group scored higher although there was no statistical significance. The effect size was large ($0.8 \leq d$).

4.4.6. *Section 4*

The results of the post-section test of section 4 are shown in the fourth level of Table 4.

On the fill-in-the-blank test, the conventional group scored higher than the proposal group, although no statistical significance was found. The effect size was medium ($0.2 \leq d < 0.5$).

On the reading test, the proposal group scored higher although there was no statistical significance. The effect size was small ($0.2 \leq d < 0.5$).

On the writing test, the scores of the proposal group were higher than those of the conventional group, although there was no statistical significance. The effect size was medium ($0.5 \leq d < 0.8$).

In total, the conventional group scored higher although there was no statistical significance. The effect size was very small ($d \leq 0.2$).

4.4.7. *Total*

The results of all four of the post-section tests are shown in the bottom level of Table 4.

On the fill-in-the-blank test, the conventional group scored higher than the proposal group, although no statistical significance was found. The effect size was medium ($0.2 \leq d < 0.5$).

However, on the reading test, the proposal group scored significantly higher in $p < 0.05$ level. The effect size was medium ($0.2 \leq d < 0.5$).

On the writing test, the proposal group scored higher than the conventional group, although there was no statistical significance. The effect size was small ($0.2 \leq d < 0.5$).

In total, the conventional group scored higher although there was no statistical significance. The effect size was small ($0.2 \leq d < 0.5$).

4.4.8. *Post-questionnaire*

In the post-questionnaire, we asked the participants to indicate which teaching method, the proposed or the conventional one is preferable for lectures on web programming. The results of the questionnaire are shown in Table 5. In questions 1, 2, and 4, most participants chose our proposed method. However, for question 3, the number of participants who chose the proposed method and the number who chose the conventional method were the same. By the binomial test, only question 1 was significant. Question 1 addressed the ease of listening to the lecture. This shows that CodEx software did help students to more easily listen to the lecture. However, our results show that CodEx software is not likely to help students to memorize the contents of the lecture.

4.5. *Considerations*

The split-attention effect was developed from the theory of cognitive load as related to learning (Ayres & Sweller, 2005). Studies have shown that integrated software reduces the split-attention effect (Cerpa *et al*., 1996), which suggests that split-attention may be caused by the instructor's need to switch back and forth among different applications.

Although the split-attention effect may also be caused by the use of a dual screen environment, some findings suggest that the use of dual screen does not always cause split-attention effect (Chang *et al.*, 2009). It is worth remarking that, as shown by Chang's study and Kuo's work, the split-attention effect can occur even with the use of a single screen (Chang *et al.*, 2009; Kuo *et al.*, 2009). Chang and Kuo studied the use of a single screen for presenting with two different applications; in our study we proposed the use of a single screen using only *one* application that is capable of fulfilling multiple instructional functions.

On the reading test, our proposal group scored higher than the conventional

Table 5. The results of the post-questionnaire.

Questions	P	C	p value by binomial test
1.Easy to listen explanation	12	4	0.011 ($p < 0.05$)
2.Understandable	10	6	0.105
3.Remember contents of lecture	8	8	0.402
4.Prefered for programming lecture	9	7	0.227

method group, except in section 1, which seems to be due to the "ceiling effect". This means that the test perhaps was too easy to show the actual difference in performance results between the two groups of learners. Although the results for some sections were not statistically significant, the total of the combined results was significant. These tests were on the grammar of JavaScript and the mechanisms of execution. Focusing on each question, the questions that asked participants to produce the correct execution results tended to show a greater effect on test scores between the participant groups. This indicates that CodEx software was useful for teaching the mechanisms of computer programming. Our results support the theory of split-attention effect as described by Ayres and Sweller (2005). As a direct implication to take into account for future developments, we should mention that understanding the mechanisms of computer programming, which involve concentrating on several sources of information simultaneously seems to require high cognitive load, which tends to increase the impact of the split-attention effect (Cerpa *et al.*, 1996).

On the *fill-in-the-blank* test, although the results were not statistically significant, the conventional method group tended to score higher. This may be because of the cognitive overload caused by viewing a lot of information on one slide at the same time. To take into account for future development, we should mention that it may be preferable to avoid the display of too much information in one single slide. Additionally, the proposed method may not be effective for memorizing the simple elements such as reserved words in respective programming languages or technical terms. Our results match the findings of Cerpa *et al.* (1996) who stated that no differences were found *on low element interactivity* tasks. They define *low element interactivity* tasks as tasks related to the *elements* that can easily be held in working memory; the effort of learning (or the cognitive load associated with learning) these elements is *low*.

On the *writing* test, our results varied for each section of the lecture. For sections 1 and 4, participants scored higher with the proposed method than they did with the conventional method, while sections 2 and 3 resulted in the reverse. This may be because the *writing* test demanded an integrated understanding of coding, therefore individuals' scores varied greatly, which increased the effect size or p value. Therefore, we cannot conclude from the results of the writing test which method is preferable. The writing test demands knowledge of both simple content such as grammatical words and complex content such as the mechanisms of programming code. This suggests that, after all, it may not be necessarily effective to teach how to *write* programming code with presentation software. Really, to acquire the ability to write code, direct practice is believed to be the best method.

Per the participants' responses to the post-questionnaire, our proposed software is beneficial when listening to an instructor explain the programming process. However, no difference was reported between the proposed software and the conventional software as to which method is more effective for memorizing necessary information. This participant feedback roughly matches our test results.

We need to point out that the small number of participants in Experiment 2 does not allow us to generalize our results. Therefore, we deem it necessary to verify our initial findings with future studies. However, although the number of participants in Experiment 2 was not large enough for generalization, our initial results are nonetheless noteworthy for the academic world.

To summarize, our proposed software is useful when teachers explain the mechanisms of programming code. However, it is not effective for rote memorization.

5. Conclusion and Future Study

5.1. *Conclusion*

5.1.1. *Development*

We have developed a presentation software program which reduces the split-attention effect on students by displaying both a source code and its execution results without the need to switch between applications.

5.1.2. *Teachers and CodEx*

We conducted an experiment to test the efficiency and usability of CodEx for teachers. Nine teachers evaluated CodEx by using the software program to give a lecture and then providing feedback on the program.

5.1.3. *Students and this software*

To test the efficiency of our software for students, we conducted another experiment. We compared our proposed teaching method with a conventional teaching method that used normal presentation software, a text editor, and a browser to display the execution results of portions of sample code.

On the post-section tests, the proposal group scored higher on the questions which measured understanding of the mechanisms of programming code. On the writing tests for code, we could not definitively determine which method is preferable because the results varied for each section.

In a post-questionnaire, we asked participants to indicate which teaching method was more appropriate for them in several learning situations. The results of the post-questionnaire corroborated the results of the post-section tests. For the question that asked which teaching method is preferable for following the instructor's explanations, participants chose our proposed program. This result may be related to the examinees' understanding of mechanisms of programming. However, participants did not prefer our proposed software for memorizing related information. This result is not contradicted to the result of the test of pure programming knowledge.

Some problems in teaching with this software were found. First, the sample codes used had to be short. This was due to the limitations in screen size. A long portion of

code cannot be shown without a scrolling function. Second, participants did, at times, experience cognitive overload caused by viewing multiple functions at the same time on a single slide. Although CodEx was useful for students in understanding the mechanisms of programming code, the test scores indicate that this software was not effective for memorizing information. This is presumably due to cognitive overload, but we do not yet have evidence to confirm this hypothesis. More research is needed on this issue.

From our results, we can conclude that CodEx is useful for students in understanding the mechanisms of computer programming codes.

5.1.4. *Advantages and disadvantages of our proposed method*

Although our proposed software offers some advantages we must also point out some disadvantages, as follows:

Among the advantages, our proposed software allows teachers to display simultaneously on just one slide both sample code and the execution results of that code. Moreover, teachers not only can show the execution results of the code, but they can also include appropriate explanation content on the same slide at the same time. Further, our findings show that this proposed software is useful for teaching the mechanisms of computer programming.

As for the program's disadvantages, this software is aimed for teaching computer programming language courses to beginners, not intermediate/advanced learners. Another issue is the added workload for teachers in preparing the slides before their lecture. Additionally, as teachers have to prepare the necessary files before preparing the slides, their teaching workload truly does increase. However, this can been seen as advantageous in that this additional preparation time up front pays off later with greater efficiency during the lecture itself.

This issue is open to future studies, including how much required time for preparing slides would be considered efficient for teachers. Further, while it is true that teachers' workload would increase with this software, it is also true that the goal of our research here is not to reduce the workload of the teacher, but to improve the usability of the software for the teacher, and perhaps more importantly, to facilitate efficient and effective learning for the student.

The teachers' *workload* was a variable that was not measured in this study as our focus was specifically on the split-attention effect on students. This does not mean that we are unaware of the issue of increasing teachers' workload, rather, we consider this issue as a topic requiring further study.

5.1.5. *Summary*

Our findings show that CodEx is useful for teaching the mechanisms of computer programming.

Although the number of participants in Experiment 2 was not large enough for generalization, our initial results are nonetheless noteworthy contributions to the academic world.

5.2. *For future study*

Development of a teaching method which accelerates the acquisition of simple information: This study revealed that our proposed software may not be useful for students in tasks involving memorization of information. This seems to be due to the increase in cognitive load when viewing multiple functions at the same time. To approach the acquisition of simple knowledge, we have to develop another teaching method or software program.

Development of a teaching method which accelerates the coding skills: In this experiment, the teachers taught "one way" lectures (without interaction or questions between teacher and students) to simplify the experiment conditions. However, in a real-world lecture, the teacher or the students can ask questions and the teacher might ask students to write sections of code as practice. It is important for students to practice writing code to acquire programming skills.

Though we have not mentioned as such in this paper, our proposed software can be used by students for their own learning, in addition to a teacher's lecture. If a teacher divides the presentation files through a network, students can execute and edit sample code on the slides. In a future study, we plan to develop, evaluate, and propose an integrated software program for teaching the language of computer programming which is more useful for the acquisition of programming skills.

Acknowledgments

The authors express their gratitude to Ms. Kathy Rice, Mrs. Elena Seleznova and Mr. Luis Inostroza for their priceless suggestions and comments, as well as their expertise and assistance while working together in checking this work and proofreading.

References

Anderson, R., Anderson, R., Simon, B., Wolfman, S. A., VanDeGrift, T., & Yasuhara, K. (2004). Experiences with a tablet PC based lecture presentation system in computer science courses. In *Proc. the 35th SIGCSE technical symposium on computer science education (SIGCSE'04)* (pp. 56–60). Norfolk, Virginia.

Ayres, P., & Sweller, J. (2005). The split-attention principle in multimedia learning. In R. E. Mayer (Ed.), *The Cambridge Handbook of Multimedia Learning* (pp. 135–158). New York: Cambridge University Press.

Buck, D., & Stucki, D. J. (2001). JKarelRobot: A case study in supporting levels of cognitive development in the computer science curriculum. In *proc. the thirty-second SIGCSE technical symposium on computer science education (SIGCSE'01)* (pp. 16–20). Charlotte, North Carolina.

Cerpa, N., Chandler, P., & Sweller, J. (1996). Some conditions under which integrated computer-based training software can facilitate learning. *Journal of Educational Computing Research*, *15*(4), 345–367.

Chandler, P., & Sweller, J. (1996). Cognitive load while learning to use a computer program. *Applied Cognitive Psychology*, *10*(2), 151–170.

Chang, Y., Chang, T., Hsu, T., & Yu, P. (2009). The impact of split-attention effect on dual-screen learning environment for programming language instruction. In *Proc. world conference on educational multimedia, hypermedia & telecommunications (Ed-Media'09)* (pp. 2110–2115). Honolulu, Hawaii.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Hillsdale: Lawrence Erlbaum.

Endo, S., & Murota, M. (2010). Development of dual-screen presentation support tool with zooming user interface. In *Proc. world conference on educational multimedia, hypermedia & telecommunications (Ed-Media'10)* (pp. 871–876). Toronto, Ontario.

Hewitt, J. (2011). *Firebug*. Retrieved January 15, 2011, from http://getfirebug.com/

Holman, D., Stojadinović, P., Karrer, T., & Borchers, J. (2006). Fly: An organic presentation tool. In *Extended abstract of conference on human factors in computing systems (CHI'06)* (pp. 56–60). Montreal, Quebec.

Kaminishi, H., & Murota, M. (2009). Development of multi-screen presentation software. In *Proc. world conference on educational multimedia, hypermedia & telecommunications (Ed-Media 2010)* (pp. 3934–3939). Honolulu, Hawaii.

Kölling, M., & Rosenberg, J. (2001). Guidelines for teaching object orientation with Java. In *Proc. the 6th annual conference on innovation and technology in computer science education (ITiCSE'01)* (pp. 33–36).

Kuo, F., Chang, T., Hsu, J., & Yu, P. (2009). The learning effects of simultaneous dual-screen instructional presentation in programming language instruction. In *Proc. the 17th International conference on computers in education (ICCE2010)* (pp. 856–863). Hong Kong.

Lanir, J., Booth, K. S., & Tang, A. (2008). Multipresenter: A presentation system for (very) large display spaces. In *Proc. the 16th ACM international conference on multimedia (MM '08)* (pp. 519–528). Vancouver, British Columbia.

Levasseur, D. G., & Sawyer, K. J. (2006). Pedagogy meets PowerPoint: A research review of the effects of computer-generated slides in the classroom. *The Review of Communication, 6*(1-2), 101–123.

Mayer, R. E. (2001). *Multimedia learning*. New York, NY: Cambridge University Press.

Mayer, R. E., & Sims, V. K. (1994). For whom is a picture worth a thousand words? Extensions of a dual-coding theory of multimedia learning. *Journal of Educational Psychology*, *86*(3), 389–401.

Nelson, L., Ichimura, S., Pedersen, E. R., & Adams, L. (1999). Palette: A paper interface for giving presentations. In *Proc. the conference on human factors in computing systems (CHI'99)* (pp. 354–361).

Raggett, D. (2005). *HTML Slidy: Slide shows in HTML and XHTML*. Retrieved January 31, 2011, from http://www.w3.org/Talks/Tools/Slidy2/

Savoy, A. (2009). Information retention from PowerPoint and traditional lectures. *Computers and Education, 52*(4), 858–867.

Sphere Research Labs. (2010). *ideone.com*. Retrieved January 15, 2011, from http://ideone.com/

Sweller, J., & Chandler, P. (1994). Why some material is difficult to learn. *Cognition and Instruction*, *12*(3), 185–233.

Tarmizi, R. A., & Sweller, J. (1988). Guidance during mathematical problem solving. *Journal of Educational Psychology*, *80*(4), 424–436.

Tufte, E. (2003). *The cognitive style of PowerPoint*. Cheshire, CT: Graphic Press.