An Evaluation Method for Panoramic Understanding of Programming by Comparison of Programmed Visual Samples

DICK MARTINEZ CALDERON [†1] YUKINOBU MIYAMOTO [†2]
MASAMI HIRABAYASHI[†3] HIDENARI KIYOMITSU [†1]
KAZUHIRO OHTSUKI[†1]

Abstract: This paper proposes enhancements to the *Programmed Visual Contents Comparison (PVCC)* method to assess Panoramic Understanding of Programming introduced in previous studies. With this method, students must compare 2 or more pictures produced by programming samples, and decide which one is more difficult to build with programming than the others, or, if the difficulty is similar for all of them. We performed previously a test with three groups of students: Game Software, IT and Graphic Design and examined the validity of the *PVCC* method by comparing the initial programming ability reported by professors of these groups with the test results. According to these results, the *PVCC* method worked well to assess programming abilities related with Panoramic Understanding of Programming. This paper proposes *PVCC* method enhancements on three topics: preparation of new samples composed by input data and output pictures, more specific ways to ask, and changes on previous samples and system..

Keywords: Computer science education, Programming Training, Student Assessment, Graphic Design, Software Engineering

1. Introduction

During the last two decades, software development has changed drastically, more and more people not involved in professional software development have become able to do programming by using new resources [1], for example: code samples and tutorials used through copy-pasting; simplified libraries, and several visual software development tools and languages, where the programming code is hidden and it can be applied with *just a click*.

As a concrete example of these changes we must refer to the inclusion of new disciplines previously considered non-related with computing, into the curriculum guidelines for undergraduate degree programs, concretely, the ones proposed by worldwide associations managing computing education standards such as: The Institute of Electrical and Electronics Engineers (IEEE) and the Association for Computing Machinery (ACM) [2].

Even when there is a significant number of external disciplines now integrated into computing education, and several studies have been proposing new systems oriented to reduce the gap between those disciplines when acquiring and applying the necessary skills on programming [3][4], an exhaustive search of the literature revealed few studies concerning the assessment of programming ability in this different range of fields, and particularly, of the multiple ways a student could understand and apply programming skills according to his knowledge level or field; in other words, how a student has (and applies) a *Panoramic Understanding of Programming*.

In a prior study we introduced the *Programmed Visual Contents Comparison Method (PVCC)* for assessment of programming abilities related with *Panoramic Understanding of Programming*. With this method, by comparing two or more displayed pictures produced by programming samples (a *Question*), a student must decide which one of the programs producing those pictures is more difficult to build with programming, or, if the difficulty is similar for all of them.

The aforementioned study reported also the application of a test to evaluate the validity of the method to groups of students of: Graphic Design, Game Design, and IT. We examined the validity by comparing the initial programming ability reported by programming teachers of these groups with the results of the test, and we found out that the proposed method worked well to find programming abilities related with a *Panoramic Understanding of Programming*.

This paper proposes enhancements to the *PVCC* Method, having as an objective to improve its effectiveness for evaluating programming abilities related with Panoramic Understanding of Programming.

These enhancements are based on feedback data obtained from programming professors of the student groups previously tested as well as professors from other universities. These professors pointed out issues and suggested changes to the method and testing system.

The proposed enhancements fall under three main topics: first, the preparation of new comparisons (*new Questions*) where 2 or more samples displaying both: input data (raw text) and output pictures are displayed, emphasizing on the difficulty of the programming processes needed to achieve the output. Second, the modification of the way to ask by including specific references to the evaluated difficulty and terms or keywords related to the assessed programming processes. Third, the improvement of previous test samples and system.

Section 2 of this text gives an overview of the *PVCC* Method and examines issues and recommended changes provided by professors, Section 3 presents the objective of the Enhancements to the *PVCC* Method, Section 4 explains the characteristics and

†1 Graduate School of Intercultural Studies, Kobe University, Kobe, Hyogo, 657-8501, Japan.
†2 Graduate School of Information Technology, Kobe Institute of Computing, Kobe, Hyogo, 650-0001, Japan.
†3 Institute of Advanced Media Arts and Sciences, Ogaki, Gifu, 503-0006, Japan.

purpose of new *Questions*, Section 5 introduces modifications to the way to ask about sample comparisons and other changes on previous samples and test system.

2. PVCC Method Basic Definition and Suggested Changes

This section provides initially, a brief overview of the basic definition of the Programmed Visual Contents Comparison Method as it was defined in previous studies [5][6], it then moves on to present the feedback obtained from professors of the four tested groups and other universities (from now on the professors) who, after looking at the results and experiencing themselves the testing system, recommended to adjust and change certain aspects of the evaluation method and the web testing system, looking towards improvement for the next test version.

2.1 PVCC Method Basic Definition

The *PVCC* method is based on the comparison of 2 displayed pictures produced by programming samples that, for our purposes we call a *Question*; if a Question is showed to the student taking the test, he is requested to decide which one of the samples is more difficult to build with programming than the other, or, if the difficulty is similar for both of them.

The correct answer to a Question for the first test was defined by a main programming structure we called *programming concept*; basically, the tested student needs to identify this concept in order to provide an answer, and the most difficult sample of each Question is based on this programming concept.

We suggested the student to answer each Question looking at it from a *programming point of view*, in other words, to think about each Question using any experience and knowledge he could have on programming, regardless of the tools or programming languages he could know. The following Questions are a good illustration of our method:

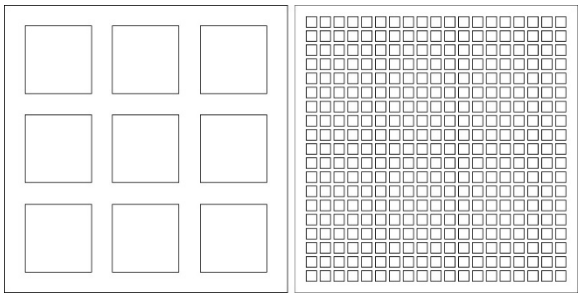

Figure 1 Question based on the *Nested Iteration* concept

The concept for the Question displayed on Figure 1 is Nested Iteration, and the correct answer for this Question would be the difficulty is similar since both samples were built by using a nested loop changing only the number of squares to be drawn.

We would expect programming experts and programmers with ability in simplified programming languages based on libraries and graphic objects only to answer that the difficulty is similar, since Nested Iteration is applied both samples.

Students with an ability limited to software tools will surely select one of both samples since they don't necessarily know about Nested Iteration therefore are most likely unable to identify the concept.

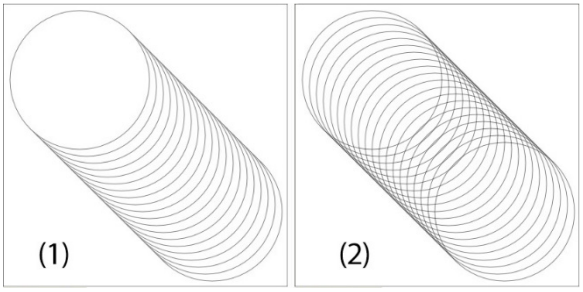

Figure 2 Question based on the *Hidden Line Removal* concept

The concept for the Question shown on Figure. 2 is *Hidden Line Removal* and its correct answer is the sample on the left marked with (1); those students answering correctly would surely know what *Hidden Line Removal* is and how it is applied on the programming sample.

Programmers whose ability is based on simplified programming languages only would answer the difficulty is similar since the same image can be obtained easily with languages like Processing by using a single function changing some of its parameters, but, they would probably misunderstand the difficulty of each sample because they certainly wouldn't know what is the content of the function applied, or what kind of algorithm is used to perform the *Hidden Line Removal*.

In the same way, people having an ability limited to graphic software tools, would probably answer based on what is shown on the pictures and could assume that both samples can be performed with the same tool on a specific software.

We built a web testing system where a set of 16 Questions was prepared for the student groups test. Figure 3 shows an example of how a Question was displayed on screen.

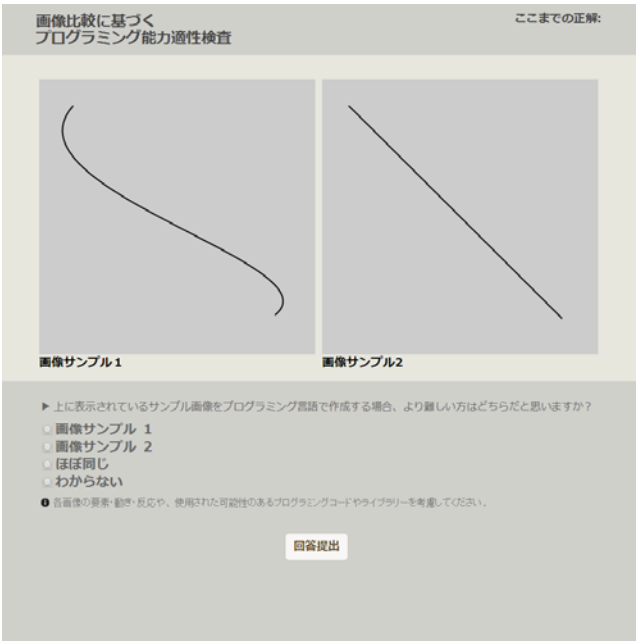

Figure 3 Example of a Question on the Web Testing System

2.2 Issues and Recommended Changes to the PVCC Method

2.2.1 Questions' Difficulty Identification

The main aspect that affected the results for all groups in the previous experiment is the identification of difficulty. Students weren't able to identify the difficulty for most of the Questions or erroneously considered the wrong degree of difficulty; this lead us to think that it is not clear enough what kind of difficulty is the one we give importance in each Question.

Samples were selected and paired considering two difficulties: the difficulty of associating images with the compared programs and the difficulty of associating the compared programs with the programming concept we wanted to evaluate for each Question, but seemingly, there were additional difficulties initially not considered, for example: the difficulty of separate the algorithms handling the interactivity, from the algorithms related with the concept to be evaluated, which could lead to students considering the degree of difficulty of interactivity algorithms more important than the difficulty of the code defining the concept we wanted to evaluate.

However, for representative Questions in the first experiment, some students were able to perceive the desired difficulty and answered correctly.

2.2.2 Revision of Programming Topics and Samples

A second aspect that needed to be thoroughly revised according to the professors' suggestions is the set of programming topics underpinning the samples used on the Questions; the initial thought was to have as a reference a somewhat wide range of programming topics found in programming books for languages such as Processing and Python [7] [8] [9] [10] [11], since these books provided an order (from beginner level to expert level) and a categorization (each chapter covered a number of subjects) similar to a curriculum, therefore we considered each of these topics as similar to a programming concept.

However, the professors noted that the referenced programming books were too focused on visual programming topics, obviating several other important subjects that are contained into curriculums for IT and Software Programming courses. As a consequence of this the majority of the test Questions are predominantly related to visual programming.

The correct answers were selected having into account the difficulty of the aforementioned programming topics according to the referenced books, in that sense we considered only one possible answer for each Question (a correct answer), thus omitting the possibility of each one of these Questions to be variously answered by people with different knowledge levels and fields, having different ideas of programs that can be harder or easier for each one of the samples.

In this sense, our criteria for choosing the set of subjects on which programming samples are based needed to be revised, having into account actual curriculum guidelines for computer-related fields, and defining what kind of answers per level and per knowledge type can emerge.

On the other hand, those Questions whose results matched the assumption for the test performed to the students are a reference of the necessary level to answer a specific Question on a specific programming subject for a specific type of people; having this in mind, we can consider to follow the same pattern these Questions have, to build and test samples not related with visual programing.

2.2.3 Adjustments on the way to perform the Question

Along with the revision to the identification of difficulty and the inclusion of programming subjects less related with visual programming, the professors pointed out that, in order to inquiry more precisely, the way to ask about each one of the comparisons needed to be improved. They argued that, probably, the Question: *which sample is more difficult to build with programming?* Is too general, and this aspect together with the already mentioned weaknesses on difficulty identification may increase the confusion the student could have, resulting in lack of certainty at answering the Question. The group of professors suggested to ask considering the relevant difficulty on each set of samples to be compared.

The suggestions presented here are addressed in the proposal for enhancements of the *Programmed Visual Contents Comparison* Method, to be detailed in the following section.

3. Enhancements of the PVCC Method

This section describes enhancements performed to the *Programmed Visual Contents Comparison (PVCC)* Method considering the suggestions provided by the professors. These enhancements includes the preparation of new Questions with additional criteria for making samples and pairing, improvement of the way to ask and changes on previous samples, Questions and testing system.

3.1 Objectives of the PVCC Method Enhancements

The main objective for this stage of the research is to improve the capability of the *PVCC* Method to assess effectively programming abilities related with a *Panoramic Understanding of Programming*.

The main issues addressed are those suggested previously by the professors, namely, a better identification of the programming samples difficulty, a more accurate classification of programming subjects based on standard curricula guidelines for computer-related courses and a better way to ask. In that sense the specific objectives for this stage are:

- To prepare new Questions with a clear difficulty and including programming topics less related with visual programming.
- To change the way to inquiry on each Question, having into account the difficulty to be evaluated.
- To propose changes to the previous samples and the test system.

The following subsections will describe, in the order already mentioned, characteristics of the new Questions and our assumption when preparing them, changes on the way to ask, and changes for the previous samples and the system.

3.2 New Questions' Characteristics

The main component for improving the *PVCC* Method is a *new type of Questions* where 2 or more samples are compared, but, instead of displaying only a graphical output, both input data (raw text, as it is inserted into the sample) and output (either picture or graph) are shown to the student.

For these *New Questions*, a Sample consists of input data and output picture(s), and a Question consists of a set of samples to be compared.

3.2.1 Samples Difficulty

The difficulty of each programming sample lies on the identification of the complexity and amount of needed processes to achieve an output from a given input data; each sample will have different programming rules and steps to transform the input data, and the output will show a representation of the input data processed according to each sample's programming rules and steps; consequently, in a sample, a single set of input data could be displayed in different outputs, and there could be many ways to input data to produce a single output.

In order to identify the difficulty of each sample and compare them, the student needs to figure out (think about) the processes through which the program transforms the input data in the displayed output(s), the student is expected to identify two aspects for each sample:

- The rules and/or steps, or the procedure (algorithm) necessary to transform the input data into the output.
- The programming structures necessary for this algorithm to do the transformation.

In this sense, the answer to a Question could vary depending on the student's knowledge; if a student, by identifying the aforementioned two aspects on each sample of a Question is able to distinguish the difficulty of one sample from another in the comparison, he most likely has different programming abilities related with *Panoramic Understanding of Programming* to those of a student who could use a different programming knowledge (e.g. based on simplified programming languages) and think of other kind procedures to transform an input data into the given output, but probably can't identify the specific programming structures contained into the algorithm, thus having another way to understand the samples' difficulty; or other student whose programming knowledge is limited to graphic software tools, who almost certainly will find it difficult to think of a programming algorithm, or programming structures.

Following the aforementioned assumption, the proposed *New Questions* fall under three categories:

- Questions with the same input data for multiple output
- Questions with multiple input data for a single output
- Questions with multiple input data for multiple output.

3.2.2 Questions with the Same Input data for Multiple Outputs

Figure 4 shows an example of a new Question with the same input data for multiple outputs. In this case there is a single set of input data: percentage of some countries population from 15-64 years; this set of data contains three categories or columns: the country name, the country region and the population percentage ordered by percentage from largest to smallest.

Figure 4 Example of a Question with the same input data for multiple outputs

For this Question there are two outputs: the first sample's output shows a pie graph with the population percentages ordered from smallest to largest starting with China, and the second sample's output shows a pie graph with the population percentages grouped and ordered alphabetically by region being the first Africa, and the first country inside this region Algeria.

The student is expected to understand that both samples contain a set of rules to do the pie graph and to parse the data for setting up the width of each circle division inside the pie according to the population percentage, but he is also expected to realize that the second sample includes additional steps to: group the countries according to the Region column of the data, organize each group alphabetically and set up the width of each division arc in the pie according to the percentage of the already grouped countries.

Students with an advanced knowledge on programming will probably be able of identifying the programming structures used by the algorithm to produce both outputs: mainly objects or arrays for country, region, and percentage, a sorting algorithm including diverse programming topics such as nested iteration and conditional statements to organize the list alphabetically according to the Region, and the countries inside each Region alphabetically, other set of mathematical processes to calculate the proportion of each percentage and transform it into an angle, and finally the use of iteration and vector drawing to graph the group of angles as a pie.

Students with knowledge limited to simplified programming languages based only on libraries and graphic objects will have criteria to say that the input data is read by a parsing function of

this library that will be used by both samples in the same way, and the outputs could be also produced by the same simplified graphing function who automatically adjusts the data to the adequate proportion in the circle and surely has an option to sort the data accordingly, so almost certainly will tend to answer that both samples have a similar difficulty.

Students with programming knowledge limited to graphic software will surely consider that the difficulty of both samples is similar since both outputs can be made without difficulty by using a combination of commands or buttons in a statistical software, or by ordering the objects manually and assigning the data directly through commands, paying little attention to the processes being carried on inside the program.

3.2.3 Questions with Multiple Input Data for a Single Output

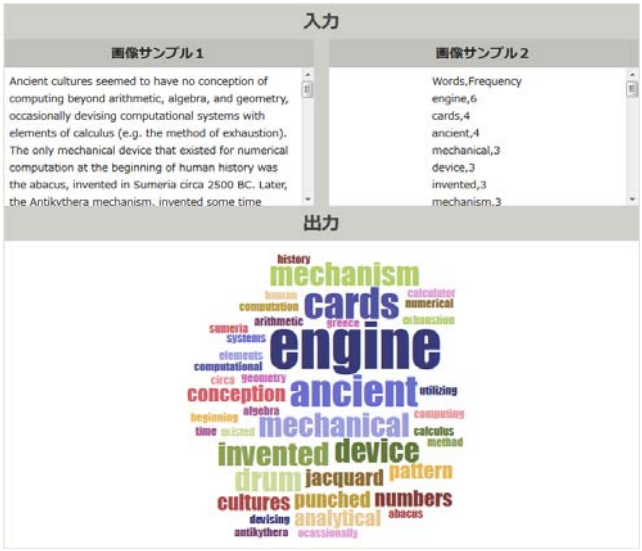

Figure 5 Example of a Question with multiple input data for a single output

Figure 5 shows an example of a new Question with multiple input data for a single output. In this case there are two sets of data: the first sample's input is raw text properly written in English, the second sample's input contains a list of the most appearing words in the text of the first sample, and the frequency of occurrence of each word. For this Question there is a single output: a word cloud composed of the most appearing words in the input text, the largest word will be the most appearing word on the list.

The student is expected to understand that, for the first sample, the raw text needs to be analyzed by the program, or specifically: the program defines a word (in English) through building a pattern to compare it to the large string of given characters that is the raw text, according to that definition the program extracts words, stores them in a list, then compares the elements of the list between each other to see if there are similarities and proceed to count how many of them are, finally it sorts the list according to the number of similarities.

This set of procedures will produce, in the end, the list on the second sample. If a student is able to realize this aspect, he will surely select the first sample as the most difficult of the set, if not, his knowledge on programming will probably be limited to simplified programming languages, so he will most likely assume that by calling the raw text file from a parsing function the text analysis is automatically done, and almost certainly will say that the difficulty of both samples is similar.

Students with programming knowledge limited to graphic software will surely consider both samples as having a similar difficulty as well, since there are several software tools to make instant word clouds by inserting raw text into a text field, and these tools have options to produce a list somewhat similar to that of the second sample.

3.2.4 Questions with Multiple Input Data for Multiple Outputs

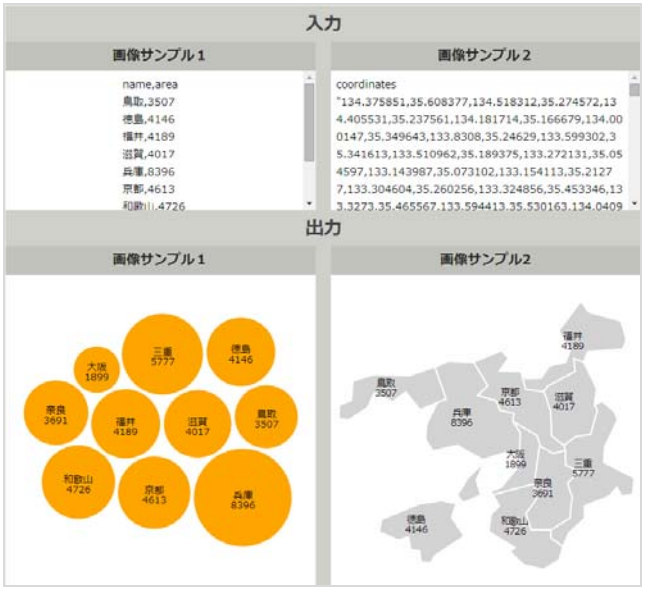

Figure 6 Example of a Question with multiple input data for multiple outputs

Figure 6 shows an example of a new Question with multiple input data for multiple outputs. In this case there are two sets of input data: the first is a list with the name and the area in Square Kilometers of each prefecture of Japan's Kansai Area. The second input contains a set of location coordinates for limit points of each prefecture of the Kansai Area. For this Question there are two outputs: the first sample's output is a group of circles where the area of each item in the input list is represented as the size of the circle, the second sample's output is a simplified map of Kansai Area with the names and areas of each prefecture correspondingly positioned.

The first impression of this Question could be that the second sample is the most difficult since it's representing both input data sources: draws a map using the coordinates of the first sample's input data and positions the information of each prefecture on the center of each prefecture's map, but, the first sample groups the circles according to a particular algorithm called Circle Packing [12]; this algorithm is of high complexity and its use to perform hierarchical data visualization is relatively new but it is being included on visualization specialized programming libraries therefore its difficulty of use has been simplified.

If a student is able to recognize that the algorithm to group the circles of the first sample's output is more difficult that the mapping of the second sample's output, even when the latter is using both data input sources, this student almost certainly has a high knowledge programming level, and the difference of Panoramic Programming Understanding ability between this student and those with programming knowledge limited to simplified languages might be significant, since the latter ones will probably tend to answer that the second sample is the most difficult, as well as those students with programming knowledge limited to graphic software.

3.3 Changes to the Way to Perform the Question

According to the suggestions provided by the professors, there is the necessity of asking about the samples in a more specific way; since there are many programming issues involved on each sample, the Question which is more difficult could be too general and may lead to confusion on the student who will probably be doubtful about which aspect of each sample must he select as the main factor to evaluate the difficulty and perform the comparison. In this sense, new Questions should make clear what difficulty aspect is being asked for and use specific keywords related with that difficulty.

On the previous subsection 3.2.1 Samples Difficulty, we mentioned that the difficulty of each programming sample lies in the identification of the complexity and amount of needed processes to achieve an output from a given input data; in this sense, a new way to ask about these samples should refer specifically to this difficulty aspect.

For example, considering Figure 5: the Question with multiple input data for a single output; a better way to ask related with its specific difficulty could be: *which set of input data makes more difficult to get the displayed output?*, since the objective of this Question is to identify which set of data makes more complex the program producing the output.

As explained earlier on the same referenced subsection, there are two aspects of each sample that the student must identify in order to perform the comparison, namely: the rules and steps (algorithm) necessary to transform the input data into the output, and the programming structures (programming topics) necessary for this algorithm to do the transformation. In this sense, a new way to ask about the samples could contain the specific terms or keywords we are looking for, namely: rules, steps, procedures, algorithm, programming structures etc.

For example, and taking Figure 4: the Question with the same input data for multiple outputs as a reference, a better way to ask about the samples could be *which sample needs more programming procedures (or steps) to get the displayed output?* Since the objective of this Question is to identify which of the graphs displayed needs more programming steps to be achieved.

3.4 Changes to Previous Questions and Samples

Together with the setting up of new samples and Questions, feedback obtained from the professors prompt us to fix several inconsistencies with previous Questions; in this sense, we consider necessary to perform the following three main changes:

1. Reclassify the set of samples into two groups: samples based on programming subjects related with visual programming, and samples based on general programming topics. To do this reclassification it is necessary to cross the information provided by curriculum guidelines regarding basic subjects in programming for all computer-related fields with the basic subjects in the referenced visual programming books. By making sure that the samples belong to a specific type of programming, we can pair them again into different Questions.
2. Remove samples containing interactivity (i.e. mouse or keyboard operation) or remove the code handling the interactivity and adjust these samples to function automatically, this to eliminate any chance of identifying the wrong programming subject and assign the wrong difficulty per sample.
3. Refactor samples where the programming subject to be evaluated is not clearly defined or where confusing elements could make a student think about different algorithms than the ones related to the programming topic we want to evaluate.

3.5 Modifications to the Web Testing System

In addition to the modifications to the method previously described, testing system usability issues were also highlighted by some of the professors, namely:

- The inconvenience of each test page's vertical layout for some screens, having into account that in some displays the user needs to scroll to reach the form and the submit button.
- Samples' long loading times that could result on system crashes and answers not being registered.
- Lack of a comprehensible instruction guide.

These issues will be addressed for the next version of the testing system in the following way:

- Changing the web testing system to a horizontal layout, to favor a visualization of the set of samples plus answering form as a whole in various displays.
- When possible, samples will be coded using web native technologies, not ports of external languages (like Processing.js, used for the first experiment samples), new Questions will be programmed using Javascript and, by removing interactivity (and in some cases, animation), loading times will be drastically reduced and errors at execution will be avoided.
- Text on the instruction guide provided at the beginning of the test will be shortened, and, having into account that interactivity will be removed, there is no need for the user to operate the mouse for anything except for clicking on the submit button.

4. Future Topics

A natural progression of this work is to perform a test using

the new Questions and compare its results to the previous test and the programming ability reported by the professors to verify the effectiveness of the proposed enhancements. Future trials of the test based on the Programmed Visual Contents Comparison Method should assess effectively the desired programming abilities.

Further studies need to be carried out in order to establish if the Programmed Visual Contents Comparison Method can effectively measure programming ability. A greater focus on establishing how to measure students specific programming abilities could produce interesting findings that account more for validate this method.

5. Conclusions

This paper set out to propose enhancements to the *Programmed Visual Contents Comparison Method*.

Issues with Questions' difficulty identification, criteria for selecting and applying programming subjects at making and paring programming samples and weakness on the way to ask about sample comparisons were addressed through the proposal of new Questions oriented to accurately discern students capable of recognizing core steps and procedures in programming samples from those who apply a Panoramic Understanding of Programming in other ways, as well as several modifications to the way to ask having in mind the specific difficulty of each type of new Questions and improvements to previous test Questions and the web testing system.

Acknowledgments We would like to thank all the Professors and Students from the College of Computing of Kobe Institute of Computing who through their participation and collaboration are making possible this research.

This work is partly supported by JSPS KAKENHI number 15K010168

References

[1] Burnett, Margaret and Myers, Brad. Future of End-user Software Engineering: Beyond the Silos. In *Proceedings of the Future of Software Engineering Conference (FOSE 2014), ACM* (New York, NY 2014), pp. 201-211.

[2] ASSOCIATION FOR COMPUTING MACHINERY (ACM); ASSOCIATION FOR INFORMATION SYSTEMS (AIS). *Curriculum Guidelines for Undergraduate Degree Programs in Information Systems (IS 2010)*. 2010.

[3] Kursat Ozenc, Fatih, Miso, Kim, Zimmerman, John, Oney, Stephen, and Myers, Brad. How to Support Designers in Getting Hold of the Immaterial Material of Software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10), ACM* (New York, NY 2010), pp. 2513-2522.

[4] Myers, Brad, Park, Sun Young, Nakano, Yoko, Mueller, Greg, and Ko, Andrew. How Designers Design and Program Interactive Behaviors. In *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '08), IEEE Computer Society* (Washington, DC 2008), pp. 177-184.

[5] Martinez Calderon, Dick, Miyamoto, Yukinobu, Kiyomitsu, Hidenari, and Ohtsuki, Kazuhiro. Difference on Visual Related Programming Understanding between Designers and Programmers by Using a Programmed Contents Comparison Method. *IPSJ SIG Notes*, 11 (2015), pp. 1-8.

[6] Martinez Calderon, Dick, Kin, Man, Kiyomitsu, Hidenari, Ohtsuki, Kazuhiro, and Miyamoto, Yukinobu. An Evaluation Method for Panoramic Understanding of Programming by Comparison with Visual Examples. In *Frontiers in Education Conference (FIE)* (El Paso, TX 2015), pp. 1-8.

[7] Bohnacker, Hartmut, Gross, Benedikt, and Laub, Julia. *Generative Design: Visualize, Program and Create with Processing*. Princeton Architectural Press, New York, NY, 2012.

[8] Shiffman, Daniel. *Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction*. Morgan Kaufmann, Burlington, MA, 2008.

[9] Shiffman, Daniel. *The Nature Of Code*. Self-published, New York, NY, 2012.

[10] Terzidis, Kostas. *Algorithms for Visual Design Using the Processing Language*. Wiley Publishing Inc., Indianapolis, IN, 2009.

[11] Lutz, Mark. *Learning Python*. O'Reilly Media Inc., Sebastopol, CA, 2009.

[12] Wang, Weixin, Wang, Hui, Dai, Guozhong, and Wang, Hongan. Visualization of Large Hierarchical Data by Circle Packing. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI'06), ACM* (Montréal, Québec 2006), pp. 517-520.