DEIM Forum 2017 P7-3

Evaluating Programming Ability by Using a Visual Contents Comparison Method

Dick MARTINEZ CALDERON† Yukinobu MIYAMOTO‡ Hidenari KIYOMITSU† Kazuhiro OHTSUKI†

† Graduate School of Intercultural Studies, Kobe University 1-2-1 Rokkodaicho, Nada-ku, Kobe, 657-8501 Japan
‡ Graduate School of Information Technology, Kobe Institute of Computing 2-2-7 Kanocho, Chuo-ku, Kobe, 650-0001 Japan

E-mail: † dick.martinez@gmail.com, † {ohtsuki, kiyomitu}@kobe-u.ac.jp ‡ miyamoto@kic.ac.jp

Abstract Until now, Paper tests and practical programming exercises have been widely used to evaluate programming abilities but in recent years, professionals in different fields have become able to do programming by using simplified software tools; as a consequence of this, they have become able to understand and do programming in a general or panoramic way. This paper proposes a new method for evaluating programming abilities based on the comparison of visual output (pictures, animations) produced by 2 or more programming samples. By comparing these output contents a student must decide which one of the programs producing them is more difficult to build with programming than the other, or, if the difficulty is similar for both of them.

Keyword Computer Science Education, Programming Training, Student Assessment, Graphic Design, Software Engineering

1. Panoramic Understanding of Programming

Software development has changed drastically during the last two decades; more and more people not involved in professional software development have become able to do programming and new resources to make programming easier have been created. For example: code samples and tutorials are being uploaded to the web and used through copy-pasting; a large number of algorithms are constantly being converted into libraries and made widely available, so to find the best-suited function within libraries has become an important task; and several visual software development tools and languages, where the programming code is hidden and it can be applied with "just a click" are being developed.

Additionally, the background and learning modes of people using programming in their fields or areas of knowledge is becoming more diverse. Nowadays Software developers and IT students learn to use code samples, libraries and interfaces as a complement to traditional ways to do programming, while students from other fields not related with software development such as arts or business are learning programming through authoring tools and simplified programming languages.

Even when many studies have been proposing new systems and software tools oriented to reduce the gap between those different fields when learning skills on programming, up to now, far too little attention has been paid to the evaluation of programming ability in this wide range of knowledge areas, and particularly, the evaluation of the different ways professionals not related with Software Development could be able to understand and apply programming skills according to their knowledge; we call this a Panoramic Understanding of Programming.

This new way to understand programming is different than the programming perception applied by software developers or programmers. To be more specific, this new understanding is not related to programming language grammar, code writing, reading or debugging, or practical performance at making programs, but instead, it's more related to an awareness of how to build a program or how to use structures or patterns in a specific programming situation without knowing in depth the foundation of those structures, or how to produce them using a specific language or to do them from scratch.

2. A Method to Identify New Abilities on Programming.

The objective of our research is to identify and measure abilities related with the aforementioned "Panoramic Understanding of Programming" in students from different fields. To perform this, we propose a Programmed Visual Contents Comparison (PVCC) method based on the comparison of 2 or more output pictures (including animations, interactive images, graphs, text) produced by programming samples, if this comparison is showed to a tested person, he is requested to decide which one of the pictures is more difficult to build with programming than the other, or, if the difficulty is similar for both of them.

The correct answer for a question is defined by the most difficult programming process (algorithm) in both samples; basically, the student needs to identify this process to provide the right answer to each question.

The person answering to any of the proposed samples comparisons is asked to think about each comparison using any experience and knowledge he could have on programming, as little as it could be, regardless of the tools or programming languages he could know. The following examples will allow us to explain more in detail this aspect:

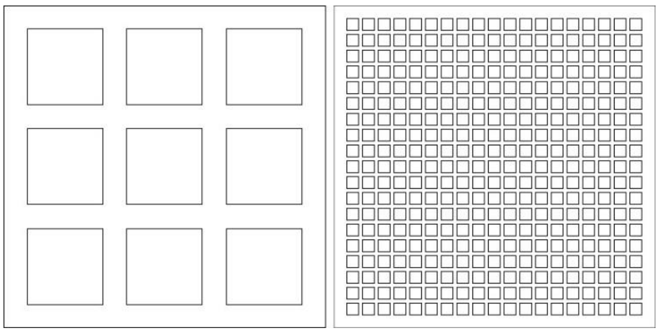


Figure 1 Comparison Including *Iteration* process

Figure 1 shows a comparison where both samples are built by using the same code (an iteration process) changing only its parameters. In this sense, the correct answer for this question was established to be: the difficulty is similar.

We would expect students who understand how the iteration process is applied on both samples to answer the difficulty is similar, since they would surely identify that both samples are built by using the same program only changing its parameters.

In the other hand, those students choosing one sample over the other as their answer are probably unaware of the specific programming process used to build both samples (iteration) and would probably consider their difficulty based more on screen presentation issues (e.g. scale, distance between objects, visual impression) than on how they are programmed.

Fig. #2 shows a Question where the sample marked with (1) uses a Hidden Line Removal process to draw circles, while the program of the sample marked with (2) doesn't use this process, therefore the correct answer for this question was decided to be: the sample marked with (1).

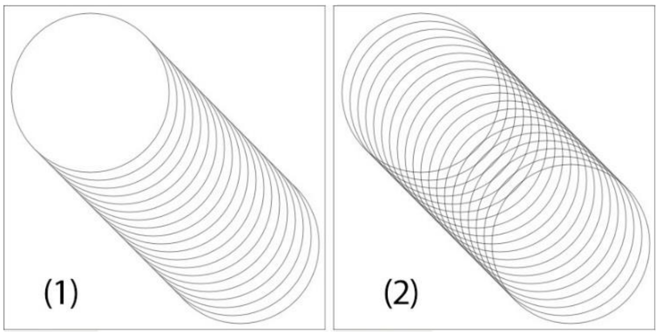
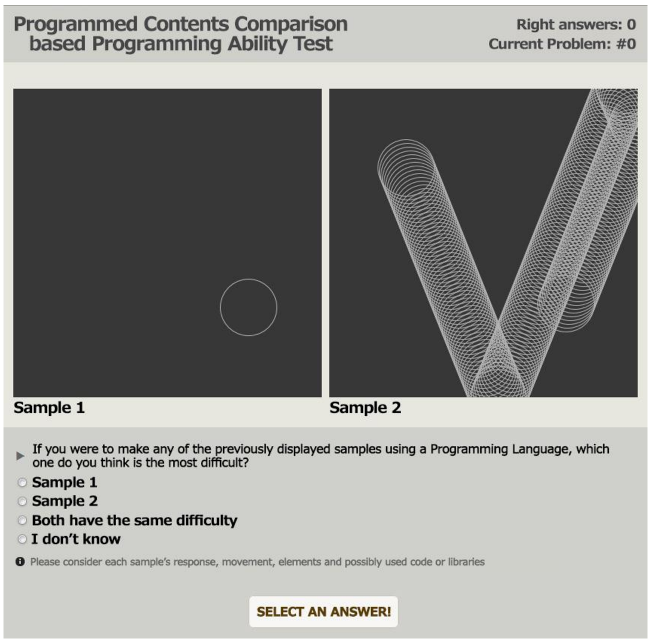


Figure 2 Comparison including *Hidden Line Removal* process

Those students of programming knowing how difficult it is to draw circles the way they are displayed on the sample marked with (1) without using any libraries, or by using older programming languages (closer to machine language), would surely understand the difficulty of the Hidden Line Removal process used on the sample marked with (1).

By contrast, those students who are used to program with simplified programming languages, or by using libraries, would probably answer that the difficulty is similar since with those languages both samples can be produced by using the same code changing only its parameters. These students are surely unaware of what kind of algorithm is the Hidden Line Removal and how it is applied.

Based on this method We built a Web Testing, where Questions including three types of samples: Static Pictures, Animated Graphics and Controlled by Mouse (sample objects can be moved or changed by hovering and clicking) were arranged.



Programmed Contents Comparison based Programming Ability Test

Right answers: 0
Current Problem: #0

Sample 1
Sample 2

If you were to make any of the previously displayed samples using a Programming Language, which one do you think is the most difficult?

- Sample 1
- Sample 2
- Both have the same difficulty
- I don't know

Please consider each sample's response, movement, elements and possibly used code or libraries

Figure 3 Example of a Question as displayed on the Web Testing System

Table 1. Feedback from the Verification Test.

Do you think this test was useful for you to know your own ability in programming?		**Do you think that by reading the explanation about the right answers and thinking the answers of this test, you have learned or reviewed anything about programming?**		**Did you find this test more enjoyable when compared with usual paper programming skills tests?**	
70%	Yes, I think so	74%	Yes, I think so	89%	Yes, I found it more enjoyable
10%	I don't think it was useful	12%	I don't think I have learned or reviewed anything about programming at all	5%	I didn't enjoy it at all
20%	I don't know	14%	I don't know	6%	I don't know

Figure 3 shows an example of how a Question was displayed on screen; in this case both samples were Animated Graphics, the sample on the left draws and erases a circle each frame, while the sample on the right draws circles each frame without erasing them.

3. Potential of the Proposed Method to Evaluate Programming Abilities.

As it was mentioned before, through using this method we want to identify abilities related to what we defined as "panoramic" understanding of programming, or in other words, a general awareness of programming through which a person can effectively make programs by putting together several external resources, without having a deep knowledge about programming (as software developers have) or having learned programming by other ways different than those applied in software development courses.

The proposed method, then, has the potential to be applied in order to measure abilities related with this general awareness of programming; for example:

- The ability to know how (and where, what library or snippet) to obtain a specific "part" or "piece" of code to make something work, depending or not of the language.
- The ability to "connect" or "replace" code parts following the logic of an already written program (perhaps written by another person).
- The ability to grasp fundamental programming structures (e.g. loops, conditional structures etc.) intuitively by understanding more complex structures (functions, objects).
- The ability to understand deep concepts of programming (e.g. resourciveness) from output elements like graphics or animations, even when not being able to do a recursive function.

In this respect, our proposed method has proven to have potential to identify the aforementioned kind of abilities.

We applied this method in a test performed with more than 200 students of different fields (reportedly: Game Design, Graphic Design and IT including software development), all of them provided feedback regarding three items which results are described in the Table 1 and will be discussed in the following paragraphs:

70% of the students indicated that they were able to measure their ability on programming by testing themselves with this method.

This result suggests that, in spite of their field or knowledge level, the students who answered this test were able to figure out how each programming sample of a comparison could be done and based on their answer they knew if they were able to handle or not a particular aspect of programming of those evaluated *(quote FIE article)*

However, their answer to the comparisons of the test even when "correct" couldn't have been appropriate from a common programming perspective, or in the opposite side, even when "wrong" could have been given according to a common programming point of view.

For example: Even though some students answering the problem of Figure 1 probably didn't have an idea of what "nested iteration" was, they were able to use a (programming) thinking different to "Nested iteration" to figure out that both samples have a somewhat similar difficulty if done with programming.

On the other hand, if fig 1 example is viewed from a too strict programming point of view, the second programming sample could be a little more difficult than the first sample. Some students probably thought on the actual difficulty required in programming to draw squares on screen and adjust them in a grid with proportional interspaces; and since the first sample has less squares it can take less time on doing it, and also require less computing resources (less processing, less memory).

This way of thinking, even when belonging probably to a person who has a deep knowledge of programming, could be considered as a "panoramic understanding" because the student is also considering a particular way to think about the program depending on the knowledge he could have.

In addition to the percentage of students that considered the test useful to know their own programming ability, a 74% of students considered that they learned a lot by knowing what the right answer was for each one of the comparisons. And interestingly a 53% of that total of students belongs to fields learning classic programming theory and how to do algorithms (namely: Game design, IT and software development).

This result could indicate that these students who know programming actually figured out that there are other different ways to do (or to think) the programs they are already used to answer, challenging their own preconception of how a program can be built, and changing the way to look for the difference between two programs.

In this sense, if a student of programming is capable of getting out of preconceived knowledge patterns and consider optimize his solution, this could also be considered as an ability related to a panoramic understanding of programming.

By comparing the test based on the proposed method with a usual programing proficiency test, namely, written tests or practical "hands-on" tests, an 88% of the evaluated students found it to be enjoyable.

This result suggests that, the whole experience of comparing two programming samples, besides giving the student the possibility to challenge their way to think about a problem (in this case a programming problem), is varied enough to provide new contents and new challenges every time and, since the comparisons are made to be answered in a fast pace (approximately 2-3 minutes per comparison), and there is no need to remember language syntax or structures.

4. Future Work.

Further studies need to be carried out in order to establish if the proposed 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 to validate this method.

Future improvement should also focus on building evaluation standards or scales for each measured ability related with Panoramic Understanding of Programming, and enhance question's classification, probably proposing different types of tests reaching different level of abilities for the same school year.

A natural progression of this work is to perform more tests using each time more complete and precise questions and keep verifying their effectiveness. Future trials of the test based on the proposed Programmed Visual Contents Comparison Method should assess effectively the desired programming abilities.

References

[1] Kursat Ozenc, F., Miso, K., Zimmerman, J., Oney, S. and Myers, B.: "How to Support Designers in Getting Hold of the Immaterial Material of Software", Proceedings of the 2010 SIGCHI Conference on Human Factors in Computing Systems (CHI '10), pp. 2513-2522 (2010)

[2] Ko, A., Myers, B. and Aung, H.H.: "Six Learning Barriers in End-User Programming Systems", Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC '04), pp. 199-206 (2004)

[3] Martinez Calderon, D., Kin, M., Kiyomitsu, H., Ohtsuki, K. and Miyamoto, Y.: "An Evaluation Method for Panoramic Understanding of Programming by Comparison with Visual Examples", Proceedings of the 2015 Frontiers in Education Conference (FIE 2015), pp.511-518 (2015)