Difference on Visual Related Programming Understanding between Designers and Programmers by Using a Programmed Contents Comparison Method

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

The main objective of this research is to look for a difference on programming understanding between Graphic Designers, Game Designers and Programmers. We propose a method whereby comparing 2 displayed images and interactive animations produced by programming samples (problems) a subject decides which one of the programs is more difficult to build with programming than the other, or, if the difficulty is similar; to solve this problems, two types of understanding are needed: one regarding the visual processing of the two pictures, and the second regarding the program making those images. The problems of this method were built considering those two types of understanding. We built a testing system based on this method and performed an experiment using this system with three groups of students: Game Software (GS), Graphic Design (GD) and IT.

1. Introduction

During the last two decades, software development has changed drastically; new resources to make programming easier have been created, therefore more people not involved in professional software development have become able to do programming, for example: several amounts of code samples and tutorials are being uploaded to the web, and any person involved in programming tasks are copy-pasting them; a large amount of algorithms are constantly being converted into libraries, or compilations of functions and made widely available, then, 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 or “black-boxed” and can be applied with “just a click” are being developed.

In addition to those changes on programming development, the background and learning modes of people using programming in their jobs or careers have diversified as well.

For example: Software developers are being taught to use code samples and libraries to do programming as a complement to the traditional “write code from scratch” traditional perspective, while graphic designers are learning programming through authoring tools and visual-based programming languages. Additionally, game designers are learning the principles of both of the mentioned professions at the same time. Considering these changes on learning modes we may assume that programming knowledge is different according to the field as well.

The objective of our research is to look for a difference on the way graphic designers, game designers and programmers understand programming in a general sense, by using their own knowledge. To look for this difference, we propose a method based on the comparison of programmed samples; with this method, from a pair of images produced by these programming samples (problems) a subject must decide which one of the programs is more difficult to build with programming than the other, or, if the difficulty is similar. We think that by using this method we can measure a “panoramic” programming knowledge, different than the knowledge related to programming language grammar, code writing and reading, or practical performance at making programs.

We built a testing system based on this method, where 16 problems were displayed, and using this system we performed an experiment with three groups of students: Game Design, Graphic Design and IT; from the College of Computing of the Kobe Institute of Computing.

On Section 2 of this paper, we make a distinction between programmers and designers making emphasis on the processes that designers carry on when programming on authoring tools, these processes define for us the panoramic understanding of programming and serve as a background to, subsequently, describe the proposed method in detail in Section 3.

Section 4 introduces and explains the characteristics of the performed experiment and Section 5 presents the results, emphasizing on findings per group and discussing representative cases.

2. Background of the Study

2.1 Designers as Programmers

In Graphic Design related courses in universities and specialized schools, it’s becoming usual to include graphic software and authoring tools programming classes, because any designer has to learn several tools combining different programming languages and management of specialized interfaces.

Programming in Graphic Design is usually taught by following a basic curriculum extracted from IT courses on the same subject, but fixing the topics according to the resources available on the tools [2], or externally in additional libraries, probably code snippets or extensions; then, when a designer deals with a new tool almost always deals with new methods to use an already learned programming language or a new programming language. Ko et al. consider that, with

[†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.

visualization tools for programming, "*learners [do] not face barriers in understanding data itself, but in trying to act on data (such as how to create or modify it)"* [3].

For example: a popular web authoring tool used by designers to program with HTML (HyperText Markup Language), CSS (Cascade Styling Sheets) and JavaScript among other programming languages is Adobe Dreamweaver [4]; this tool has several snippets and pre-made objects that can be dragged into a visible template of a web page, and supposedly will work at execution time but sometimes they don't. Besides, there are many "tricks" to make appear specific pieces of code or to control diverse processes; these tricks usually require a long sequence of mouse clicks, searching around the tool menus or the interface and dealing with programming code directly; Adobe Inc. (creators of Dreamweaver) announced a new web design tool called Muse that supposedly allows graphic designers to: *"create unique, standards-based websites – without writing a single line of code"* [5], with this new tool Graphic Designers need to get used to new methods or "tricks" to be able to use the same programming languages.

Because of this double-tool handling and considering that a graphic designer is an end-user programmer, or a person who needs to use programming in his projects but is not entirely dedicated to that [1]; he will consequently solve programming problems recursively: by trial and error, by pulling in and taking out those code snippets and pre-made objects, by possibly trying out a few lines of code he could have found online; in the end by "sketching" code [1][6], that he himself has built without having any idea if it's optimal or standard-compliant, not even how many hidden bugs it will have, through this "sketching" of code this designer is applying his design knowledge.

This "sketching" way of programming will eventually bring difficulties if intended to be performed in a professional level, because most of the times its learning curve is shallow, leading to what Ko et al. call "Simplifying Assumptions" [3]. By "sketching" code a designer is constantly learning from diverse sources, and trying diverse ways to solve a programming problem that, in the end, summarizes in a product he assumes is good but, when compared with what it should be, many mistakes can emerge, resulting in "Knowledge Breakdowns"[3]; in other words: if a program is bad, surely all the methods applied by the designer to do this product will fall down too, and he will need to start learning again having the same possibility of making those simplifying assumptions and having knowledge breakdowns again.

2.2 Particularities of Graphic Designers' Programming Understanding

Even when it could be considered an unappropriated way to do programming, "sketching" code allows graphic designers to transform the knowledge they acquire on programming, an external matter, into something more related with their visual nature; so they generate a new kind of programming knowledge merged with design concepts [1][2]; different from programming learnt through formal, academic ways on IT and software development fields.

In any design project, a graphic designer needs to establish a connection, primarily visual, with any material he needs to use or handle; he expects some of the skills he assumes as fundamental related with the appropriation of objects (for instance: drawing, diagramming, getting to know physical characteristics like: color, form, texture) to be available when programming; Ozenc et al refer to this as the "immateriality of software" [6]; so through sketching a designer tries to bring materiality to code.

Referring in detail to the process a designer performs, Ozenc et al. mention that:

"*In their work to envision 'what might be', designers engage in reflection in action (discovering the idea at the point of rendering it) and reflection on action (stepping back to assess what they have made as they plan their next move)*" [6].

Designers need scenarios, stages, where to explore the relation between objects (things), space, and environmental factors, but they find really difficult to do this with code; they lack the understanding of "code" as "objects" because they don't have anything that holds up their perception.

Norman refers to this process as "Affordance", that is: *"A relationship between the properties of an object and the capabilities of the agent that determine just how the object could possibly be used"* [7].

Programmers on the other hand only gets to know and become aware of software graphic elements when they deal with subjects like: User Interface Programming, Web development, or Multimedia (Sound or Video). But unlike designers, they don't do programming by "seeing" but by "reading", they manage languages so they need to be aware of syntax, coherence and particularly, errors; they focus on code patterns and coding style to "catch the bug".

Regarding this aspect LaToza and Myers state that:

"In coding activities, developers select among various strategies to answer the questions necessary to complete their tasks (...). When exploring code, developers seek information, make decisions about which structural relationship to traverse to find information" [8].

Programmers then, make relationships between structures instead of objects, they cannot perceive things like color, or shape but they decipher code and, to do that, they apply "strategies" instead of "sketching".

3. Programmed Contents Comparison Method

The objective of our research is to look for a difference on the way graphic designers, game designers and programmers understand programming in a general or panoramic sense, by using their knowledge.

To seek this difference we propose a "*Programmed Contents Comparison Method"*. With this method, by comparing 2 displayed images and interactive animations produced by programming samples, a subject decides which one of the programs producing those images is more difficult to build with programming than the other, or, if the difficulty is similar for both of them. For our purposes, a set of two programming samples will be called a "problem".

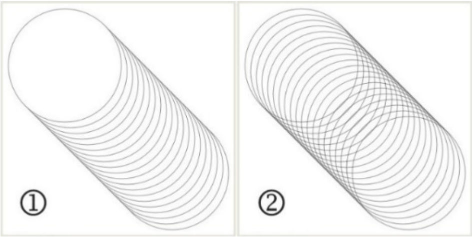


Figure 1 Example of a problem (a pair of samples)

Figure 1 shows an example of the proposed problems. The concept for this problem is *"Hidden Line Removal"*, the same image can be obtained easily with graphic authoring tools, as well as with diverse programming languages based on graphic objects libraries (like Processing), but even when ① case seems a simple change (the color filling of the circles is extremely easy to achieve in any graphic tool), the programming concept here involved: *"Hidden Line Removal"* is more complicated than the one involved on ②. The answer to this problem will depend on the person's association of the images with programs and the identification of the *"Hidden Line Removal"* concept.

By applying the "*Programmed Contents Comparison Method"*, we think we can see if a subject is capable of solving programming problems without thinking about programming language's syntax, code writing and reading; instead, by using his own understanding about how the programming of those samples works and by applying his own knowledge and way of thinking.

If a subject is capable of: associate the compared images to the programs producing them, grasp (perceive, see) their main structures, understand which one of those structures is more difficult and from there give an answer to one of the programming problems proposed; we can say that this person was able to solve this programming problem if he gives the correct answer.

Having into account that the programming problems on our method are based on programming structures that designers and programmers learn and are used to deal with; if designers and programmers provide distinct answers, we think that the difference on their understanding of each structure in general could become evident.

3.1 Problems Preparation

3.1.1 Problem Selection and Classification

Each problem is based on the difficulty level of one sample over the other regarding a main programming structure we called: "programming concept"; basically, the subject needs to identify this programming concept in order to answer correctly; this concept is the base of the most difficult sample of each problem.

To build a prototype of the method, several programming concepts were summarized from programming books oriented to designers and developers; having as a main selection criteria its level (beginners to experts) and considering if this concept is representative of programming in general, or if along those books the concept is studied by graphic and game designers, as well as by developers and programmers [9][10][11][12][13]; the chosen concepts were, respectively: Bezier Line, Nested Iteration, Coordinates Storage and Recalling, Erasing and re-drawing, Boundary detection, Easing, Timer, Area delimitation, New position according to previous position, Change through time, Animation using trigonometry, Picture Pixel Management, Recursion, Lists, Empty Area Recognition and Hidden Line Removal. In total 16 problems were prepared.

3.1.2 Problems' Degree of Difficulty

We designed the whole set of problems to have two kinds of difficulty for each one: first, the difficulty of associate images with programs; to surpass this difficulty, we consider that the subject answering the problems probably needs to:

- Understand what is each sample doing (how it is moving, what is happening) by looking at the images on screen.
- Identify what elements is each program using to do what it's doing (for instance: if there is a circle on the picture, there probably will be a circle on program, if there is a vertical movement on the animation, there probably will be a vertical movement on the program).
- Understand how the objects the program is using are working together to give that (visual) result (for example how a circle is connected with the movement it's doing or the position and timing it's appearing).

Second, the difficulty of associate the programs with the programming concept; to surpass this difficulty we think the subject probably needs to:

- Think about, and/or recall from his own knowledge and/or experience:
  - What kind of programming structure can be used to achieve this movement, or effect? (For example: the subject could be asking to himself: how a circle moving on screen can be moving it? through what programming structure or concept?)
  - What is the main effect of each of those structures? (The subject probably asks himself: if we apply that structure to something, what is the result? And, is that result coherent with some of what is currently happening on the pictures?).
  - How many programming structures or concepts he can apply into the objects appearing on the screen, and how many ways of application does they have (alternative uses)
- Identify which is the main concept for each sample (what is the more relevant programming concept?).
- Compare both main concepts, for both samples.

Following this line of thought, we arranged some problems to have a difficult image-program association, that will need more knowledge on images or graphic software tools management; some other problems were thought to have an easily identifiable image-program association but the comparison between programs and the connection with the concept will need a deeper knowledge on programming; and finally, we thought problems with both characteristics, where both kinds of knowledge will be needed.

It doesn't matter if the concept is thought to be applied through a library or an internal simplified function of a programming language, neither if applied with a graphic tool; the purpose of each problem is for the student to be able to think about the concept using his own understanding of programming as general and different as it could be (depending on the field).

4. Experiment

In this section we describe the characteristics of the experiment performed using a web testing system with three groups of students.

4.1 Programmed Contents Comparison Testing System

We built a web testing system based on the described method, where the set of 16 problems was displayed. Problem's contents were developed using Processing.js and in order to make the system compatible with modern browsers, the interface and database were developed using web current technologies and programming languages such as: JavaScript, MySQL and PHP.

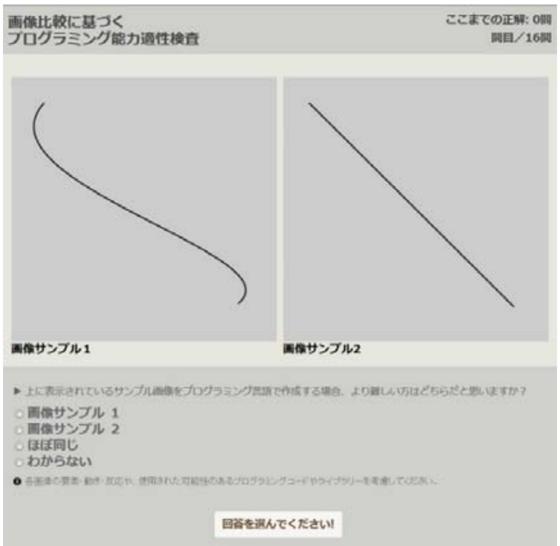
Having into account that the difficulty is to be evaluated from the programming contents, the answering method was built to be simple and straightforward, having a unique question: *"which sample (of the pair displayed on each problem) is more difficult?"* and four answer options: "*The first sample", "The second sample", "Both of them" and "I don't know"*. During the test the student must choose only one answer within those options, then click on a "submit" button to store his answer on a database and pass to the next problem. The test was thought to be carried on sequentially (one problem after another) and in one try, the subject was asked to answer all and each one of the problems and the time needed to answer one problem was considered to be 30 sec. to 1 min.

Table 1 shows the displaying order of the problems in the test, and the programming concept of each numbered problem.

Table 1 Programming concept for each problem number

Problem Number	Programming Concept
#1	Bezier Line
#2	Nested Iteration
#3	Coordinates Storage and Recalling
#4	Erasing and re-drawing
#5	Boundary detection
#6	Easing
#7	Timer
#8	Area delimitation
#9	New position according to previous position
#10	Change through time
#11	Animation using trigonometry
#12	Picture Pixel Management
#13	Recursion
#14	Lists
#15	Empty Area Recognition
#16	Hidden Line Removal

Since there is only 1 question and 4 answer options across all the 16 problems, the answers are compiled on the database assigning a number to each one; therefore, for the original question: *"Which sample (of the pair displayed on each problem) is more difficult?"* if the answer is stored on the database as "0", that means this person answered *"sample 1"*, if the answer is stored as "1" this person answered *"sample 2"*, and so on.



画像比較に基づく
プログラミング能力適性検査

ここまでの正解: 0問
問目／16問

▶ 上に表示されているサンプル画像をプログラミング言語で作成する場合、より難しい方はどちらだと思いますか？

○ 画像サンプル 1
○ 画像サンプル 2
○ ほぼ同じ
○ わからない

● 各画像の要素・動き・反応や、使用された可能性のあるプログラミングコードやライブラリーを考慮してください。

Figure. 2 Example of a problem on the Web Testing System

On Figure 2 we can see the appearance of a problem when seen on screen; the screen is divided to contain a first section (header) with information of the current amount of correct answers and the number of the current displayed problem; a second section (contents) that includes the two programming samples to compare; and the third section (question) displaying the question with the list of answers to be chosen using radio-buttons and the submit button.

In addition, by the end of the test, a complete report with user's answers per question compared with their respective correct answers and a brief explanation about the evaluated programming concept is displayed; likewise, each subject has the opportunity to answer a brief questionnaire regarding the whole test experience.

4.2 Student Groups Characteristics

The experiment was conducted with three groups of students from the College of Computing of Kobe Institute of Computing: The first group was Graphic Design (GD) on the 1st year integrated by 32 students; their curriculum includes subjects where Graphic Software Tools for Photo Edition, Illustration, Desktop Publishing and 3D Modeling are taught together with Web Coding and Web Design, and only programming languages oriented to Web (HTML, CSS, JavaScript) are studied.

The second group was IT and Software (IT) in the 2nd year integrated by 41 students; their curriculum includes subjects where programming languages such as: C, Java and Assembler are taught, also Algorithm Theory is studied together with Web back-end programming and networking. This curriculum doesn't include classes where any graphic software tool or visual-related programming language has to be used or studied.

The third group was Game Software (GS) on the 3rd year integrated by 61 students; Besides of Game Design related subjects such as: Graphic Design Principles, Character Design, 3D Modeling and Animation; their curriculum includes subjects where the same programming languages studied by IT are taught. Additionally, this curriculum includes subjects on graphic libraries for those languages (for example: DirectX on C++), Web Coding, Algorithm Theory and Mathematics. Graphic Software Tools are used mostly on Game Design classes.

Professors in charge of the three groups reported their scores obtained on a previous paper-based programming ability test; these scores followed the pattern: **GS>IT>GD**, in other words: Game Software (GS) group achieved the best score on the test, followed by the IT and Software group (IT), and in the last position was the Graphic Design (GD) group.

5. Results

In this section we examine the process through which we look for difference on the experiment results, followed by a discussion about problems that obtained a significant difference (Representative Problems) dividing them according to the group who obtained the highest score on each; this discussion considers the possible reasons for each of these problems to be advantageous for a particular group.

The amount of answers per option per problem were compared with the correct answer for each problem to obtain the amount of correct answers per group for each problem and for the whole test per student.

Being unequal groups, we had to establish the percentage of correct answers per problem for each one of the groups, **Table 2** shows the percentage of the total of correct answers and average for each problem per group, highlighting problems with high and low scores.

Table 2 Percentage of correct answers per problem highlighting problems with high and low scores per group

Problem Number	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**#12**	**#13**	**#14**	**#15**	**#16**	**Avg**
Game Software % correct answers	▨ 90	▨ 74	64	🟧 31	62	▨ 85	54	41	▨ 67	🟧 5	🟧 26	▨ 69	▨ 84	▨ 72	44	51	**57**
Graphic Design % correct answers	▨ 88	🟧 31	▨ 72	44	▨ 88	▨ 81	44	🟧 34	▨ 72	🟧 19	🟧 16	▨ 66	53	63	41	🟧 25	**52**
IT and software % correct answers	63	▨ 66	56	37	▨ 71	▨ 80	41	37	▨ 66	🟧 20	🟧 20	39	63	49	🟧 27	41	**48**

Conventions ▨ High Score 🟧 Low Score

By using the correct answers percentages, we could establish difference per problems between the three groups by comparing: GD with IT and; GD with GS and GS with IT.

Having the differences on the correct answers for each group we could see which problems had a significant difference on its correct answers' percentage; considering these problems as representative we performed an F-test of equality of variances over the original results according to the groups' comparison previously mentioned, according to the results of this test, for each of the compared sets of data we performed a two tailed T-test to confirm the validity of the difference for each representative problem.

Table 3 shows the difference on correct answers' percentage between the groups highlighting the representative problems, or the problems that had a significant difference for, at least one of the performed comparisons.

Table 3 Significant difference on percentage of correct answers verified through T-test per group comparison per problem, highlighting representative problems

Problem Number	▨ **#1**	▨ **#2**	**#3**	**#4**	▨ **#5**	**#6**	**#7**	**#8**	**#9**	▨ **#10**	**#11**	▨ **#12**	▨ **#13**	▨ **#14**	**#15**	▨ **#16**
% Difference GD vs IT	🔵	🟠	✕	✕	🔵	✕	✕	✕	✕	✕	✕	🔵	✕	✕	✕	🟠
% Difference GS vs IT	🟡	✕	✕	✕	✕	✕	✕	✕	✕	🟠	✕	🟡	🟡	🟡	✕	✕
% Difference GD vs GS	✕	🟡	✕	✕	🔵	✕	✕	✕	✕	🔵	✕	✕	🟡	✕	✕	🟡

Conventions
▨ Representative Problem
🔵 Significant Difference Advantaging GD
🟡 Significant Difference Advantaging GS
✕ Problems without Significant Difference
🟠 Significant Difference Advantaging IT

5.1 Representative Problems Having a Difference Advantaging GD

GD obtained a comparatively better average per problem than IT, particularly on problems having easier image-program association and a difficult comparison and programming concept association, but was overtaken in most of the problems by GS' score.

This group obtained a result somewhat lower than GS but higher than IT on two problems, namely: "Bezier Line" (#1) and "Picture Pixel Management" (#12), had the best score of the three groups on one of the representative problems: "Boundary Detection" (#5) and almost the same result than IT on "Change through time" (#10) problem.

5.1.1 Bezier Line (#1)

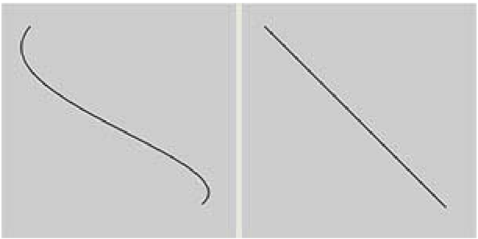


Figure 3 Appearance of "Bezier Line" problem

This problem belongs to the group requiring more knowledge on images management or graphic tools; it contains static samples, therefore interaction wasn't needed.

IT obtained the lowest percentage of correct answers (63%), and the GS and GD obtained almost equal percentages of correct answers (90 % and 88% respectively).

This problem was thought to have a really simple set of images to interpret, but the concept is more familiar for GS and GD because several authoring and creation tools are based on this kind of graphics (in fact, one of the first concepts to learn when dealing with those authoring tools is the difference between a "Pixel based image" and a "Bezier based image" and the complexity of the last one). given that either: a curve line or an straight line can be written in many programming languages by using only one function (sometimes the same function with different parameters) regardless of if it's Bezier or not, some of the IT students could have thought that the difficulty was similar.

5.1.2 Boundary Detection (#5)

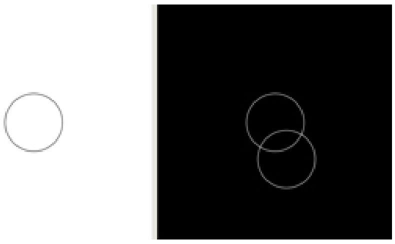


Figure 4 Appearance of "Boundary Detection" problem

This problem belongs to the group requiring both, knowledge on programming and on images management or graphic tools and it requires mouse interaction; on the first sample, if the mouse pointer hovers over the circle, the background turns black; for the second sample the mouse pointer is replaced by a circle so if the static circle is intersected by the moving circle, the background turns black.

GS obtained the lowest percentage of correct answers (62%) while GD and IT groups obtained 88% and 71% respectively.

For GS the "Boundary detection" concept is fundamental, either for programming and screen design. Games are based on the detection or not of objects' limits in order to perform actions, and in order to create environments (worlds, terrains) boundary detection is needed as well. By looking at GS' data on other answers for this problem, the answer "both of them" is on second place; 17 of 61 people thought this problem was of similar difficulty. From the point of view of a game designer probably both samples had the same difficulty because they seemed to be related to boundary detection; a graphic designer on the other hand is probably used to see the first sample when dealing with interface button behaviors, available as easily changeable options in several authoring tools while the second sample shows an action caused by the (precise) intersection of two forms, since the second one considers the area and the point of intersection of the two circles, it is more difficult for GD to imagine in programming.

5.1.3 Change Through Time (#10)

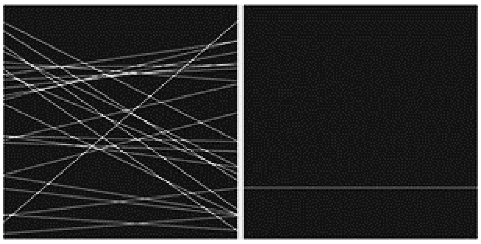


Figure 5 Appearance of "Change Through Time" problem

This problem belongs to the group requiring both, knowledge on programming and on images management or graphic tools; it includes animated samples, but interaction isn't needed.

For this problem IT obtained 20% of correct answers while GD obtained 19% and GS obtained 5% but, we need to consider that from the total population, considering all the three groups only 17 students of 134 answered correctly.

We think that this problem needs to be revised in order to see why did it perform poorly, but considering the rest of the answers apart from the correct ones, the majority on the three groups was inclined to answer that the first sample is the most difficult; we think there are some possible reasons for this situation, the first is: sample # 1 seems visually complicated; and the second reason is: apparently, a more difficult concept than the one we wanted to evaluate was included in the first sample: "line position according to coordinates". In other words, in the second sample the position of the starting point and ending point of the line on the X axis is the same on each step while the line moves on the Y axis sequentially in one direction only; but in the first sample, the starting and ending point of each line on each step as well as the position on Y axis change randomly, these are not sequential.

5.1.4 Picture Pixel Management (#12)

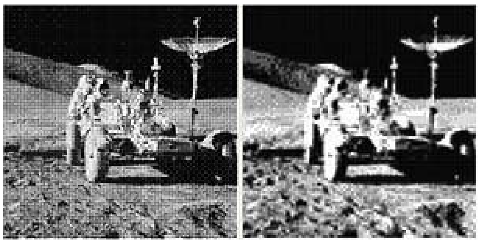


Figure 6 Appearance of "Picture Pixel Management" problem

This problem belongs to the group requiring more knowledge on images management or graphic tools; it requires mouse interaction as well; the first sample changes the brightness of a picture according to the position of the mouse while the second sample replaces every pixel by a circle according to the depth of color of a base picture, and changes the size of each circle according to the position of the mouse.

GS and GD obtained 69% and 66% respectively while IT obtained 39% of correct answers.

The results of this problem were expected because this problem involved Image Processing knowledge, and pixel management is a basic concept for both GD and GS, these groups of students understand the concept of Pixel from their first years of their careers.

5.2 Representative Problems Having a Difference Advantaging IT

According to the results, in general, IT performed comparatively worse than GD and GS on most of the problems but had a better result in two of them, namely "Nested Iteration" (#2) and "Hidden Line Removal" (#16); and one requiring both kinds of knowledge: "Change through time" (#10).

We have already analyzed the "Change through time" problem on the previous subsection about GD results, having this into account we will discuss here the remaining two problems.

5.2.1 Nested Iteration (#2)

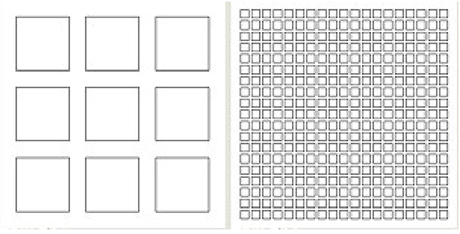


Figure 7 Appearance of "Nested Iteration" Problem

This problem belongs to the group requiring more knowledge on programming; it contains static samples, therefore interaction isn't needed.

GD obtained the lowest percentage (31%) being one of lowest of the group's whole test; comparatively IT and GS obtained comparatively high percentages (66% and 74% respectively).

The concept "Nested Iteration" is basic in programming for matrix allocation, basic search through lists and matrices, among other procedures; both samples on the problem included that concept; the fact that this problem was answered correctly by most of IT students shows us that probably they have certain knowledge on the graphical representation of the "Nested Iteration", therefore they were able to surpass the difficulty Image-Program, while graphic designers only get to see this kind of graphics through authoring tools. By looking at the second sample GD found this one more difficult to perform probably because of the steps needed to achieve it using a graphic tool, having into account its amount of graphic elements (squares).

5.2.2 Hidden Line Removal (#16)

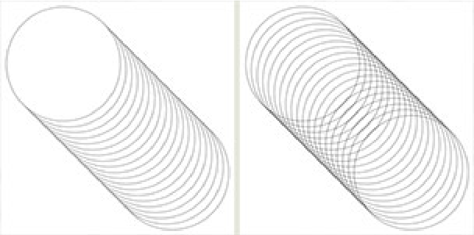


Figure 8 Appearance of "Hidden Line Removal" Problem

This problem belongs to the group requiring more knowledge on programming; it contains static samples, therefore interaction isn't needed.

The difference is higher for GS and IT, whom obtained 51% and 41% of correct answers respectively while Graphic Design group obtained only a 25% of correct answers.

Apart from the correct answers, the rest of the answers for this problem show that the majority of students of GD were inclined to think that sample #2 was more difficult. As we explained previously on section 3.1 of this paper, we expected the "Hidden Line Removal" process to be identified; this wasn't achieved by GD who surely lacks the knowledge related with this main programming concept. We expected GD to, at least, identify the difficulty as similar for both samples since, in order to get the same pictures with graphic tools, a simple process of changing the filling of the ovals with commands is to be performed, but probably the "visual disorder" of the second sample tricked them.

5.3 Representative Problems Having a Difference Advantaging GS

GS achieved the highest percentage of right answers for most of the problems, this group was able to surpass both difficulties regardless of what kind of problem was presented; as we thought, they possibly perform very well when dealing with authoring tools and visual-related programming languages.

GS had the best score of the three groups on six of eight representative problems, namely: "Bezier Line" (#1), "Nested Iteration" (#2), "Picture Pixel Management" (#12), "Recursion" (#13), "Lists" (#14), and "Hidden Line Removal" (#16). We have already analyzed "Bezier Line", "Nested Iteration", "Picture Pixel Management" and "Hidden Line Removal" on the previous subsections, therefore we will discuss here the remaining two problems.

5.3.1 Recursion (#13)

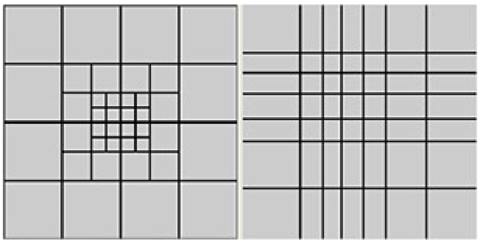


Figure 9 Appearance of "Recursion" problem

This problem belongs to the group requiring both, knowledge on programming and on images management or graphic tools; it requires mouse interaction as well, when the user clicks on each sample there is a change: for sample #1 this change follows a recursive algorithm, the second sample only draws two crossing lines in the position where the click is performed.

The difference is higher for GS, who obtained 84% while IT and GD obtained 63% and 53% respectively.

This problem is of high complexity in both the visual part and the programming part, even though the sample on the right could become visually disorganized as the clicks' number raises, the sample on the left looked much more symmetric. A GD Student could be familiar with the concept of "fractal", at least to having read about, or seen visual samples of what a fractal "looks like"; what we want to highlight here is the fact that GS obtained a comparatively high score, and the difference with GD is also high; from this result we can say that probably GS has a solid knowledge of this concept in a programming level; and this problem allows us identify a particular understanding of programming from GD that seems to be solid from the Image Management side.

We may think that IT could be more familiar with the fractal concept at a programming level than with its graphic representation; we could also think that they weren't able to associate the second sample to a simpler program.

5.3.2 Lists (#14)

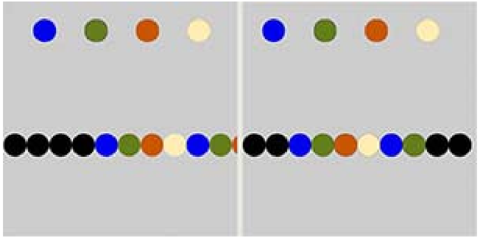


Figure 10 Appearance of “Lists” problem

This problem belongs to the group requiring more knowledge on programming and it needs mouse interaction as well, for the sample #1, when the user clicks on each colored circle a new circle is added to the end of the “array” of black circles, while on sample #2 the color circle is added on the middle of the “array”.

The difference advantages GS, who obtained 72% while IT and GD obtained 49% and 63% respectively.

“Lists Management” concept is fundamental in programming and, has a high difficulty. The result advantaging GD and GS could mean that, for programmers familiarized with lists management (IT’s case), both samples are equally easier; we verified this by looking at the other answers for this problem in this group where “both of them” received the second larger amount of answers.

6. Limitations and Future Topics

6.1 Limitations

This method was implemented as a prototype, in this sense, only results from three test groups were obtained; through the analysis of the results provided by the application of this method to the mentioned groups we were able to obtain only:

- Enough evidence to say that there is a difference on the three groups’ understanding of the programming problems included.
- Particularities of this difference per problem and per group, including: best and worst answered problems by group, difference in correct answers per problem per group, and characteristics of the difference on answers.

6.2 Future Topics

In further stages of this research, we want to enhance the testing system to make it capable to identify and measure programming abilities; we are considering to make use of this system to measure how much of (in what degree) a specific programming ability does a student have or is able to apply to solve a programming problem as well as to make it useful to be applied on different expertise levels and for identify curricula-specific abilities by field; not only on Graphic Design, Game Design and Software Development fields but with other kind of professionals using programming in their daily jobs.

7. Conclusions

We were able to use the proposed method to perform the comparison of programmed contents with the purpose of looking for difference. This method’s comparison can be performed in the future with other kind of samples and other kind of programming concepts to obtain more results regarding the found difference.

Results indicate as well that this method was useful to find a difference on programming understanding between graphic designers, game designers and programmers. Additionally, results from the performed questionnaire showed a positive feedback regarding the test system. Students recognized that this test allowed them to evaluate their own ability on programming; besides, by comparing this test with a usual paper based ability test, they thought this test to be more enjoyable.

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 made possible the realization of this project’s experiment.

References

[1] Myers, B., Park, S.Y., Nakano, Y., Mueller, G. and Ko, A.: How Designers Design and Program Interactive Behaviors, Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '08), pp. 177-184 (2008).

[2] Park, S.Y., Myers, B. and Ko, A.: Designers' Natural Descriptions of Interactive Behaviors, Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '08), pp. 185-188 (2008).

[3] 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).

[4] Adobe Dreamweaver CC Website Builder, available from < http://www.adobe.com/products/dreamweaver.html>.

[5] Adobe Muse Web Design Software, available from < http://www.adobe.com/products/muse.html>.

[6] 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 SIGCHI Conference on Human Factors in Computing Systems (CHI '10), pp. 2513-2522 (2010).

[7] Norman, D.: The Design of Everyday Things. Basic Books, New York (2013).

[8] LaToza, T. and Myers, B.: On the Importance of Understanding the Strategies that Developers Use, CHASE '10 Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering, pp. 72-75 (2010).

[9] Bohnacker, H., Gross, B. and Laub, J.: Generative Design: Visualize, Program and Create with Processing. Princeton Architectural Press, New York (2012).

[10] Shiffman, D.: The Nature Of Code. Self-published, New York (2012).

[11] Shiffman, D.: Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction. Morgan Kaufmann, Burlington (2008).

[12] Terzidis, K.: Algorithms for Visual Design Using the Processing Language. Wiley Publishing, Inc, Indianapolis (2009).

[13] Lutz, M.: Learning Python. O'Reilly Media Inc., Sebastopol (2009).

ⓒ2015 Information Processing Society of Japan