



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

MARTINEZ CALDERON DICK ORLANDO

(Degree)

博士 (学術)

(Date of Degree)

2018-03-25

(Date of Publication)

2019-03-01

(Resource Type)

doctoral thesis

(Report Number)

甲第7235号

(URL)

<https://hdl.handle.net/20.500.14094/D1007235>

※ 当コンテンツは神戸大学の学術成果です。無断複製・不正使用等を禁じます。著作権法で認められている範囲内で、適切にご利用ください。



博士論文

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

(視覚コンテンツ比較によるプログラミング能力評価法)

平成 30 年 1 月

神戸大学大学院国際文化学研究科

MARTINEZ CALDERON DICK ORLANDO

Table of Contents

1. Introduction	1
1.1. Background	1
1.1.1. Characteristics of Nowadays Programmers.....	2
1.1.2. What Kind of Programming Skills Do Nowadays Programmers Learn?.....	6
1.1.3. Can Current Methods of Evaluation of Programming Abilities Evaluate Nowadays Programmers' Skills?	9
1.2. Research Objective and Stages.....	13
1.3. Structure of This Thesis.....	15
2. Related Research	17
2.1. Fundamental Research on Evaluation of Programming Abilities	17
2.2. Research Related to Alternative Programming Methods.	18
3. Proposal of a Programmed Visual Contents Comparison (PVCC) Method to Evaluate Abilities related with a Panoramic Understanding of Programming	20
3.1. What is Panoramic Understanding of Programming?	20
3.2. What Programming Skills do we Want to Evaluate?	22
3.3. How to Evaluate Skills Related with a Panoramic Understanding of Programming?....	25
3.3.1. Why Including Programmed Visual Contents?	26
3.3.2. Why Comparing Programmed Visual Contents?	27
3.4. The Programmed Visual Contents Comparison Method (PVCC).....	29
3.4.1. Problems' Degree of Difficulty	30
3.4.2. Programmed Visual Contents Comparison Testing System.....	32
4. Verification of PVCC Method's Potential to Evaluate the Programming Ability of Novice Programmers.	34
4.1. Objective and Context of This Chapter	34
4.2. Procedures to verify the PVCC Method's Potential to Evaluate Programming Skills of Novice Programmers	35
4.2.1. Participants' Characteristics and Previous Programming Ability	35
4.2.2. Assumption Based on Reported Programming Ability	36
4.2.3. Results and Comparison to Find Significant Problems	37
4.3. How Well Can the PVCC Method Evaluate Novices' Programming Ability?	38
4.3.1. Problems whose Results Match the Assumption.....	39
4.3.2. Problems whose Results Mismatch the Assumption Useful to Evaluate Programming Ability.....	40

4.3.3.	Problems whose Results Mismatch the Assumption Useless to Evaluate Programming Ability.....	43
4.3.4.	Non-representative Problems	44
4.4.	Conclusion of This Chapter.....	44
5.	Extending the Range of Programming Skills Assessable by The PVCC Method	46
5.1.	Objective and Context of This Chapter	46
5.2.	New Problems' Characteristics	47
5.2.1.	How Input Data Can Guide the Answer: Problems with Same Input data for Multiple Outputs.....	48
5.2.2.	How Data is Needed to Understand the Program: Problems with Multiple Input Data for a Single Output.....	51
5.2.3.	How the Absence of Data Can Indicate Difficulty: Problems with Multiple Input Data for Multiple Output.....	53
5.3.	A Classification for Programming Processes	56
5.4.	Test Oriented to Professors to Verify the Suitability of New Problems.....	57
5.5.	Are New Problems Suitable for Evaluating Programming Abilities Related to a Panoramic Understanding of Programming?	58
5.6.	Conclusion of This Chapter.....	60
6.	How to Create Suitable Problems for the PVCC method?	61
6.1.	Identification of the Most Difficult Programming Technique.....	61
6.2.	Balance of Complexity and Concision of Problems.....	63
6.3.	Revision of Programming Topics and Samples	65
7.	Verifying PVCC Method's Potential to Evaluate the Programming Ability of IT Experts in Relation to Programming Experience and Knowledge.	66
7.1.	Objective of This Chapter	66
7.2.	Procedures to verify PVCC Method's Potential to Evaluate IT Expert's Programming Abilities in Relation to Programming Knowledge and Experience.....	67
7.2.1.	Consolidation of Modified Problems.	67
7.2.2.	Questionnaire and Guidance on Experience and Knowledge of Programming and Software Tools.	69
7.2.3.	Participants Characteristics Including Previous Experience and Knowledge.	72
7.2.4.	Assumption Based on Reported Experience and Knowledge	73
7.3.	Is It Possible to Evaluate IT Experts' Programming Abilities in Relation to Their Experience and Knowledge by Using the PVCC Method?	74
7.4.	Why do IT Experts Fail at Answering Problems Based on the PVCC Method?.....	76
7.5.	What Kind of Programming Knowledge and Experience Do IT Experts Have?	79
7.5.1.	Programming Loop.....	81

7.5.2.	Recursion.....	83
7.5.3.	Empty Area Detection	85
7.5.4.	Distribute Objects According to Data	87
7.5.5.	Visualization According to Data	89
7.5.6.	Data Format Reading.....	91
7.6.	Conclusion of This Chapter.....	93
8.	Conclusions	95
	References.....	97
	Acknowledgements	100
	Appendixes	101
	Articles Product of this Research	132
	Refereed Articles.....	132
	Non Refereed Articles	132

1. Introduction

1.1. Background

Over the past 20 years, worldwide institutions who establish the standards for computing education have been suggesting the inclusion of diverse disciplines from knowledge areas previously considered as *external to computing*, into curricula proposals.

For example, The Curriculum Guidelines for Undergraduate Degree Programs in Information Systems (2010) proposed by the Association for Computing Machinery (ACM) and the Association for Information Systems (AIS), presents a relation of the coverage of different fundamental and elective proposed courses across the 17 sub-disciplines of Information Systems (IS), going from Application Developer, passing by Business Manager and finishing in disciplines like: User Interface Designer and Web Content Manager. This document lists up the fundamental skills needed by professionals in IS according to his emphasis, making clear which courses are or are not essential for some of those disciplines. [1]

This is only an example of the transformation of disciplines from other fields (such as Management, Business and Arts) into computing-related subjects. Other examples can be found in Curriculum Guidelines published by these institutions for other areas of knowledge such as: Information Technology, Software Engineering and Computer Science.

In some of the courses included in the guidelines, particularly those related with Web Content Management, User Interface Design and Human-Computer Interaction, it is usual to include programming classes together with graphic software tools instruction. Programming for these fields is commonly taught by using visual programming languages and graphic tools where fundamental topics are transformed and adjusted according to the resources available on those tools [2] [3], or in additional libraries, probably code snippets or extensions.

Recently, researchers have examined the consequences of changes on the structure of Information Technologies' education and the impact the new ways to do programming have in methods to teach and assess the skills a student need to perform successfully in an Information Systems related professional environment. Andrew Ko *et al.* make a relevant question regarding this issue:

“Now, an increasing number of (...) programmers control manufacturing robots, create spreadsheets, and design interactive prototypes. Yet, for such growth to occur, millions of aspiring (...) programmers must overcome substantial learning barriers in programming systems. Do we know enough about these barriers to design systems that help these individuals?” [4]

1.1.1. Characteristics of Nowadays Programmers.

In the same way many disciplines previously not belonging to the Information Technology world are now included into computing-related curriculums, nowadays there has been a significant increase of programming jobs performed by programmers who lack formal training in computer science and software development, may not be professional software developers, but instead are professionals in other disciplines like the mentioned in the previous section, users of software produced by others, that have become able to design and produce their own programs.

The software that these programmers produce is initially thought for personal or limited use, mainly made to solve problems or add functionalities to bigger software platforms, but after a time their software can become popular or available for wider audiences via internet or as the main product of a small enterprise.

Margaret Burnett and Brad Myers report, as a part of their analysis on the future of End-User Software Engineering that,

“More than 12 million people in the U.S. say they do programming at work, (...) compared to only about 3 million professional programmers (...). Clearly then, end-user programming empowers (...) [or] it has already empowered millions of end users to create their own software”. [2]

Later in the same report they mention that users of software become programmers to accomplish tasks their own way in these software platforms. Professionals in aforementioned disciplines like: Management, Business, and Art are in fact doing programming to solve their own field-related problems when using software.

Burnett and Myers also report that many subpopulations of programmers lack training on formal computer science, so it can be said that a professional in other field can become a programmer without knowing how to do (formal or standard) programming.

This is the main characteristic of nowadays programmers: the lack of formal training in software development; but even though this could imply that the gap between

programmers not formally-educated and professional developers is significantly large, and that there is no reason to delegate a *professional* programming task or entire job to someone who only has a superficial knowledge in software development nor to rely on his final result, most of those not formally educated programmers ultimately find their products being published and actually used frequently by people. [2]

An example of this contrast can be found in the world of Web Design and the recent changes derived from the inclusion of formal programming aspects into previously considered non-programming tasks or jobs: HTML (Hypertext Markup Language) and CSS (Cascade Styling Sheets), both structural languages to build web pages were considered until recently not programming languages but “[*technologies to*] *define and describe the content of a web page*” [5] or “*simple mechanisms for adding style (...) to web documents*” [6], and designers using these technologies to make web pages were not considered programmers; nowadays thanks to the introduction of compilers, complementary languages for CSS and automatic generation of HTML templates through engines among other extensions; web design has become a programmers’ job and those designers doing it need to learn how to program properly with these languages before tackling the creation of a *well-made* website.

As a consequence of this and other changes on the context where they performed their job, web designers became programmers and their field gained differentiation but added a greater responsibility as well; anyone can do a web page, but now, a *well-made* web page can be created only by someone with the specific skills required to manage the aforementioned new programming extensions to HTML and CSS, together with programming languages that add functionality to those web pages; nowadays designers must become proficient in all those programming tools.

The situation is similar in fields like Statistics, Economics, and (Media) Art, where changes in technologies that involved the introduction of programming have been tackled with two strategies we already mentioned: one, the addition of (basic) programming lessons into curriculums [1], and two, the creation of tools to support the lack of programming and software development basic knowledge [7].

The second main characteristic of nowadays programmers is the kind of programmers they end up becoming, as Burnett and Myers mention:

“A premise of classical software engineering research is that professional developers have some training and/or experience in software engineering methods, and

also at least a little interest in following good software engineering practices. In contrast, end user software engineering research cannot assume the presence of any of these” [2]

While professional developers are educated in the standard procedures for producing good software, those programmers that lack formal training consider other standards according to the field they are professionals in, and adjust or accommodate these standards into programming to be able to obtain what they want, as Ko *et al* argue, this is a difference on *intent* rather than on experience,

“An end user is simply any computer user. We then define end-user programming as programming to achieve the result of a program primarily for personal, rather [than] public use (...). In these (...) programming situations, the program is a means to an end and only one of potentially many tools that could be used (...). In contrast (...), professional programming has the goal of producing code for others to use. The intent might be to make money”. [8]

An interesting point to highlight here is that, even when the final product of a development project made by non-educated programmers is supposedly intended for personal use, this product can become popular (or widely used) after a time, therefore the aforementioned *intent* changes.

It is assumed that these kinds of programs will be used only by their creators, but in many cases these products are considered solid and robust enough by not only a few people but thousands or millions who can download and use them, or in the other hand, a programmer who has created programs for personal use only, could start pondering that his experience doing programming is enough to offer services related to these skills, so he can be more attractive for job offerings and potential clients.

Since the learning curve for (formal) programming (and software development) is quite steep, a non-professional programmer would first consider if he can do what is required for his product to be suitable with the tools he knows as a professional in other field, and then he would resort to new tools depending on his (field of) knowledge, not necessarily (and not primarily) programming languages but tools that help him overcome his lack of programming knowledge; these tools can go from complete software suites that allow to do programming products by only inserting objects via button click inside an interface, to programming libraries complementing languages that allow the programmer to simplify programming complex tasks.

As a consequence of this, the non-professional programmer who wants to make code for others becomes a developer without the necessary knowledge nor experience to be a professional developer, but carrying on with the full responsibility of this role; and as mentioned, his *intent* changes radically. Burnett and Myers point out the consequences of this situation:

“Evidence abounds of the pervasiveness of errors in the software that end users create (...). Even when errors in end-user-created software are non-catastrophic, however, their effects can matter”. [2]

The third characteristic of nowadays programmers is the way how they learn to solve problems in programming; the way they learn and the knowledge they acquire is specific not only to the field they belong to but also to the task (or barrier) they try to accomplish (or overcome).

Ko, *et al* in their study regarding barriers in end-user programming systems provide a list of six general barriers that programming learners should surmount every time they try to perform programming tasks. Namely: *“Design, selection, coordination, use, understanding, and information barriers”* [4]

According to the definition provided by Ko *et al*, these barriers are fundamentally points during the learning process where the programming learner having to learn a new programming technique, after considering its learning curve makes simplifying assumptions regarding the environment, programming language and tools (e.g. libraries) he is using to try to learn how to achieve this task; sometimes (and as Ko *et al* report, most of the times) these assumptions are invalid, when this happens the student has *knowledge breakdowns*, or moments where he finds that all the time and effort invested in acquiring new specific knowledge was wasted because when applied, this knowledge didn't work as expected, thus having to start again, adding more steepness to the learning curve, or interrupting completely the learning process.

It is interesting to note that these programmers coming from other fields most likely make these simplifying assumptions having in mind the way those problems or tasks are solved in their respective fields of expertise, so they make assumptions of the type: *I thought I know how to do it but I didn't know how to make it work or it didn't work as expected*.

1.1.2. What Kind of Programming Skills Do Nowadays Programmers Learn?

As discussed above, nowadays programmers know how to design programs according to their own *intent*, and when they learn programming they choose the tools and resources that suit their skills considering this intent.

Let us now think again about two of the six general learning barriers that Ko *et al* defined previously, because their definition will allow us to understand how nowadays programmers learn programming and what kind of specific programming abilities they develop by applying field-specific experience and knowledge into the development process.

Ko *et al* explain that *Design Barriers* are,

“Inherent cognitive difficulties of a programming problem, separate from the notation used to represent a solution (i.e. words, diagrams, code)” [4]

According to the authors' explanation, in their study programmers were not able to *visualize* optimal solutions for several programming problems including among others: sorting elements, programmatic communication between forms and conditional logic.

If we assume that nowadays programmers having these sorts of problems when learning, according to the software tool they use, can solve them by using different ways to mix objects (like forms, or sort boxes) that they can see; It is likely that instead of having the ability to abstract the programming process and conceive an optimal way to put it into working code, they have the skill of understanding the nature of the problem *visually* first and answer to it by building their own *affordance* for it.

An affordance according to the definition provided by designer Don Norman 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”* [9] that for this case could become: a syntactical view of the relationship between the options available in the tools they know and their own capabilities (knowledge, experience) that can determine just how these options can be used to build the specific object (program).

An example of this affordance can be found in the workflow of Visual programmers, or those programmers who create programs (patches) in visual programming tools like Max MSP [10], VVVV [11] or EyesWeb [12]; these

programmers are commonly professionals in media art, sound design, animation and other related fields.

Patches are essentially diagrams that become programs in real time; those diagrams are a visual syntactic representation of the program that indicates the flow of data and connections between objects and black-boxed processes. If they were to create for example: a registration form that communicates with another external form they would diagram the form fields first, then connect them, and confirm the data flow inside the fields and with the outer form *visually*.

This ability of *thinking programming visually* is also commented by Ko *et al* when comparing a program to a factory and proposing their *factory methaphor* oriented to software designers, which is quite similar to what was mentioned already for visual programmers.

“A program is a factory and the learner is a factory’s creator. Factories are made of machines (programming interfaces) which are coordinated to systematically receive, manipulate, and produce products (program output and behavior). In general, learners have many tools to help create, run and inspect their factory (the programming environment)”. [4]

Abstract operations with objects that software developers can understand like: Pulling and pushing data, are better understood visually by the aforementioned visual programmers.

Another significant type of learning barriers mentioned by Ko *et al* are the *information barriers*, which are defined as:

“Properties of an environment that make it difficult to acquire information about a program’s internal behavior (i.e. a variable value, what calls what)”. [4]

These barriers emerge when programmers *guess* how a program can behave but they can’t check the hidden behavior.

Going back to Max MSP, VVVV and EyesWeb, these tools were build considering *guessing* as an important part of their way to do programming, therefore, they are able to show the flow of data managed by a program in real time and they implement easier ways to visualize this flow.

Programmers working with these tools are able to change this data flow in real time by, for example, interrupting the connection, inserting a box containing another process that changes the orientation, type or movement of the flow, divide into various

other flows going other ways, or recirculating the flow through separate sub-patches that will transform it. The importance of this method of doing programming is that the definitive or correct way to re-flow the data is not defined, the programmer *guesses* according to his intent in the very moment what he wants this flow to do, there is no previous pattern for doing it correctly.

Guessing and *visualizing* are programming abilities acquired commonly by artists designers and sound engineers who work with different media; several (if not most of the) software tools targeted to these professionals contain programming platforms similar to those mentioned as a complement of common interfaces.

Even though by using and learning programming with visual programming tools or other tools that help with the process of understanding code programmers who are not professional developers can *create* (to some extent) his own flow of data and his own design, and the degrees of freedom these tools allow make easier the application of opportunistic and emergent development strategies like *Guessing* and *Visualizing*; it is clear that many standard development steps that formal programmers are taught to follow, memorize and make intrinsic part of every development project are sacrificed in benefit of creative power, as Loksa and Ko *et al.* mention:

“There are many technologies designed to teach coding in more engaging ways (...). However, studies of these learning technologies show that rather than use them to learn to code, most learners primarily use them to create content, possibly avoiding coding altogether.” [13]

If we consider that coding could be considered the main task of programming, the aforementioned argument could seem rather ironic, and this is one of the main arguments against calling *programming* to what some of nowadays programmers do with the aforementioned tools and considering their products *programs* but, on the other hand, through applying opportunistic strategies by using software tools to simplify programming, these programmers can acquire much more faster abilities that only expert developers demonstrate.

Let us use some words from the study of LaToza *et al* that describe the proficiency of expert developers regarding some programming abilities like: debugging and generation of multiple approaches to problems, in order to explain the similarity between these abilities and those that an end-user programmer can demonstrate. According to LaToza *et al*:

“Experts debug faster by generating better hypothesis while studying less code (...) Experts select from multiple strategies for accomplishing tasks, are capable of generating multiple alternatives before making a choice”. [14]

First, programmers using emergent and opportunistic strategies like the aforementioned *guessing* and *visualizing* can generate better hypothesis over their own design while studying less code. Through *visualizing*, these programmers acquire the ability of looking over their own programs’ design and data flow, this makes them aware of their own flaws and capable of solving them faster.

Second: through *visualizing* and *guessing*, with the help of the appropriate tools, programmers can solve programming tasks by using different ways to mix objects that they can see or interact with, have the capacity of grasping the nature of the problem *visually* first, to then build their own affordance of it.

Having established what are the characteristics of nowadays programmers; and exploring how these programmers learn to program and what kind of abilities do they develop, it is important to examine if current evaluation methods can assess these abilities.

1.1.3. Can Current Methods of Evaluation of Programming Abilities Evaluate Nowadays Programmers’ Skills?

Traditionally, one of the most used and considered solid methods of evaluating programming skills has been to put them to test by solving practical code challenges. It has been also argued that it is also possible to improve an individual’s own programming skills through challenging him with proficiency or performance tests; to this respect Loksa *et al.* consider that:

“Coding involves skills that go well beyond how to use a language (...) the more complex a programming task is, the more that both novice and expert programmers exhibit metacognitive self-regulation behaviors”. [13]

During the last decade and, together with the popularity of online education and the already mentioned involvement of design disciplines into computing and programming, many coding challenge websites have appeared that argue to be capable to *evaluate* individual (or grupal, even institutional) coding skills while *teaching* how to code, by challenging programmers to code proficiency or practical tests by using several methods.

Abundant examples of these programming skills evaluation/teaching platforms can be found across the web, one of the most representative could be *codewars.com* [15], a site that, by following a training schema that resembles those used in martial arts, including what they call *katas* (the correct way to apply a technique), claims to be *better than college* at teaching programming techniques covering an ample variety of languages: from the popular JavaScript and Ruby, to the classic C and C++.

Another example is the widely known *Project Euler* [16], a collection of programming problems that started in 2007 with more or less a hundred and fifty problems and ten years later has collected more than six hundred problems.

Since its beginning, problems on this site were limited to computational mathematics but in the last years Project Euler has extended his range to cover a more general spectrum of programming challenges, the complexity and times each problem was solved since its inception is indicated across the list of problems, and the more problems an individual solve the highest his rank is. Probably one of its most prolific contributor could be *Project Nayuki* [17] who boasts having acquired the 8th level by solving more than five hundred problems from this site, in various programming languages; all of them are published in its own site.

An additional example could be *HackerRank* [18], a site that besides evaluating and teaching programming through code challenges, also serves as a recruiting site; by using the results of code challenges and other indicators they claim to be able to match every developer with the right job.

Many other similar sites with more or less the same structure than the aforementioned ones can be found, among others: The *Aizu Online Judge* [19] which contains problems from All Japan High school programming contest; *TopCoder.com* [20] that besides programming also evaluates and teach data science and UX design skills, and, *exercism.io* [21] that contains a significant collection of programming problems, particularly for languages only used in numerical analysis and hard computing science like: Emacs Lisp, Haskell, Julia and Fortran.

Thomas LaToza and Brad Myers report on their study regarding the importance of understanding the strategies developers use that, not until recently, evaluation studies for the activities of software developers started identifying goals, needs, questions and strategies used by developers rather than focusing on testing only performance at building

specific programs or knowledge about programming patterns or syntaxes; LaToza and Myers argue that,

“In coding activities, developers select among various strategies to answer the questions necessary to complete their tasks (e.g., fix a bug, implement a feature). These questions and hypotheses about answers form a hierarchy, as developers decompose questions into lower-level questions that are easier to answer with the available methods and tools”. [22]

Unfortunately, neither the proficiency tests nor the perspective presented by the aforementioned authors seem to be applicable when talking about most of nowadays programmers, particularly those who lack education in formal development since, as it was already mentioned, their programming activities are identified as emergent, and the strategies they use as unplanned and opportunistic, as Ko *et al* argue,

“Because end-user programmers’ designs tend to be emergent, like their requirements, requirements and design (...) are rarely separate activities. This is reflected in most design approaches that have been targeted at end-user programmers, which largely aim to support evolutionary and exploratory prototyping rather than upfront design”. [8]

If we look subjectively programming from the perspective of Ko *et al* and the type of programming challenges that the aforementioned test platforms promote, tools like: Max MSP, VVVV or EyesWeb probably don’t support upfront program design, nor allow formal programming; therefore those programs done by using these tools may not be fitting to *good design* principles, and neither will they be accepted solutions to any code challenge proposed by sites like Project Euler or exercism.io, even if by using these tools and applying emergent or opportunistic development strategies those challenges can be solved.

While there is an extensive body of research related to the creation of systems to support programming and learning activities of nowadays programmers with the aforementioned characteristics, particularly oriented to make these activities comply with formal development standards and patterns, there is little interest on systems to assess or evaluate how the alternative strategies that these programmers apply by mixing their field-specific knowledge with programming affect (positively) the way to create programs and also the final product.

Considering the case of professional designers as an example, while the principles of design are applied the same way to create products, there is still a strict difference between *software designers* and *other designers*, and those tools created to *support* the creation of end-user programs are also *enforcing good behavior* at creating them, as Ko *et al* suggest that,

“An alternative to enforcing good behavior is to let end-users work in the way they are used to working, but inject good design decisions into their existing practices. One crucial difference between trained software engineers’ and end users’ approaches to problem solving is the extent to which they can anticipate design constraints of a solution” [8]

One major drawback with this approach is that considers the way *other designers* (not software designers) design as *bad behavior*, this approach assumes in a general sense that a programmer whose main background is *design* in other field different than software design cannot anticipate correctly *design constraints*, which for the case of *other designers* may be an incorrect assumption.

Noteworthy attempts to change the point of view on how non-developer programmers can propose alternative ways to think programming problems by using (conceptual and practical) resources from different fields can be found in many studies like the research carried on by Mangnano, LaToza *et al* who analyzed the role of designers’ type and style of whiteboard sketches in presenting alternative approaches to program design, [23] and illustrates thoroughly how alternative design strategies and management of resources (visual, syntactical, written) can lead to better interaction and communication between designers and software developers at the design stage and more effective ways to represent software design problems and their possible solutions.

There are also several studies on *opportunistic programming* among which stand out those of Brandt *et al.*; they have described the advantages of *opportunistic programming*, or:

“The activity of building non-trivial software systems with little to no upfront planning about implementation details, and ease and speed of development are prioritized over code robustness and maintainability” [24].

when applied in several phases of the development cycle like: design and debugging.

These authors have also highlighted the integral use of web information foraging when performing opportunistic end-user software development. In their research they describe how non-developers learn programming techniques by searching examples across the web, applying them to their own programs and enhancing their own applications by looking out for more examples and interleaving them with their own code; even if those examples are complex, those programmers are able to apply basic principles of development like *divide and conquer*, sometimes without having acquired this knowledge in any course; to this respect they argue that:

“Programmers use web tutorials for just-in-time learning, gaining high-level conceptual knowledge when they need it. (...) [they] deliberately choose not to remember complicated syntax. Instead, they use the Web as external memory that can be accessed when needed” [25].

1.2. Research Objective and Stages

Considering the aforementioned issues, the main purpose of this research is to propose a method to evaluate programming skills related with the alternative understanding of programming generated particularly by programmers who lack formal training in computer science and software development, may not be professional software developers, but instead are professionals in other disciplines as discussed in chapter 1 (for the purposes of this thesis all of them will be called *nowadays programmers* from now on). We call this alternative understanding a: *Panoramic Understanding of Programming (PUP)* (for details regarding this term see section 3.1).

With the *Programed Visual Contents Comparison (PVCC)* Method, a programmer is presented with a comparison of two or more pictures produced by programming samples, then, he must decide which one of the programs producing each one of these pictures is more difficult to build with programming than the other, or, if the difficulty is similar. (for details regarding the PVCC method see section 3.3 and 3.4).

This research consists of three stages with different objectives for each one of them: The first stage involves the application of the PVCC method on a programming ability test performed with novice programmers belonging to different fields such as: Graphic Design, Game Design and Information Technologies (IT). During this stage, the validity of the PVCC method for evaluating programming ability on novice programmers

was verified by comparing the initial programming ability reported by programming instructors and teachers of these groups with the results of the test.

Analysis on the results of this test showed that the PVCC method worked well to find programming understanding abilities on the novice programmers groups tested, but there were suggestions and discussion points proposed by their instructors and professors that became the base for the following stage of the research.

For the second stage our objective was to extend the range of programming abilities that the PVCC Method can evaluate. We aimed to accomplish this by adding input data to output pictures and focus strictly on the programming processes needed to obtain these pictures from the provided input data. The main focus of this stage is the preparation of new test problems where two or more samples displaying both: input data and output pictures are shown.

During this stage, we performed a test with programming professors of different universities to verify the suitability of the new problems. The analysis of the results of this test allowed us to determine if the new problems were appropriate or not to evaluate programming abilities and what was lacking and or missing in each one of them.

Based on the analysis of the results and suggestions on adjustments and changes on the new problems provided by professors on both tests, we include a chapter on what points need to be considered when building adequate test problems for evaluating *PUP* by applying the PVCC method.

The objective for the third stage is to verify if it is possible to evaluate the programming ability of expert programmers with the PVCC method. More specifically, we want to know if we can evaluate the programming ability related with PUP of IT (Information Technology) experts who give an account of their own experience and knowledge in programming.

For this stage we consolidate selected problems from previous first and second test, corrected according to the suggestions provided by professors on how to create adequate questions for the PVCC method. In addition to the samples comparison each problem includes an explanation on what kind of programming techniques we are focusing in, and a questionnaire on three topics related with the programming technique in question:

- 1) *Awareness or perceptiveness of the main programming techniques*: for these questions the person answering the problem clarifies his understanding on the explanation provided and acknowledges if he or she would have realized about the use of the process without reading the explanation.
- 2) *Experience on handling similar techniques on software tools*: for these questions the person describes his previous experience with similar techniques on software tools (not programming languages), for each problem we provide an example of what kind of software tools could include similar techniques.
- 3) *Experience dealing with the programming technique* in question and more complex code or algorithms that include this technique by using any programming language.

We apply the aforementioned selection and combination of problems and questionnaire in a new programming ability test performed with IT professionals (in development, infrastructure and systems design and operation). The validity of the PVCC method for evaluating programming ability in relation to these experts' knowledge and experience in programming techniques was verified by comparing their reported experience and knowledge with their results at answering problems with sample comparisons.

The analysis on the results of this test shows that the PVCC method works well to find programming understanding abilities in relation with experience and knowledge for the group of IT experts, with new discussion points that will become the base for further advances on this research.

1.3. Structure of This Thesis

Having the stages of this research above mentioned in mind this report is organized in the following way:

Chapter 2 introduces previous research related, Chapter 3 is concerned with the *Programmed Visual Contents Comparison* Method, Chapter 4 addresses the verification of the validity of the PVCC method to assess novice programmers' programming ability, including the analysis on the results of a test carried on with 4 groups of novice

programmers from different fields, and a discussion on how well this method can evaluate the programming ability of these novice programmers.

Chapter 5 examines how we extended the range of programming abilities the PVCC method can evaluate by creating new questions; this chapter includes the results of a test performed with programming professors to verify the suitability of new problems.

Chapter 6 discusses how to build appropriate problems for the PVCC method according to feedback and suggestions obtained from professors and instructors in previous tests.

Chapter 7 presents the work performed in order to verify if the programming ability of IT experts in relation with their knowledge and experience can be evaluated with the PVCC method.

Finally, Chapter 8 concludes this thesis by summarizing the objective of this research, the solid points and issues of each one of the tests performed based on their results and how these issues were addressed according to expert's feedback and results' analysis; also, this chapter highlights positive feedback obtained from programming learners and participants regarding the enjoyability of the tests, and presents further work planned for this research.

2. Related Research

There has been an increasing amount of literature on assessment of programming skills during the past two decades, which is mainly research performed with students of computer science and software development, but thanks to an early interest on bringing software and programming literacy to schools as a way to develop higher cognitive skills some of these studies have been focused in assessing programming thinking and performance in fields not related directly with computer science. Then, as a result of the rapid changes in the context of the aforementioned external disciplines now involved in the IT world, several studies derived from this fundamental research are focusing on the creation of supporting tools and methods to fill the gap between professional and end-user software development.

2.1. Fundamental Research on Evaluation of Programming Abilities

One of the initial academic products of the aforementioned interest in bringing programming to schools could have been the work of Roy D. Pea and D. Midian Kurland. “*A Study of the development of programming ability and thinking skills in high school students*” [26] where the authors report a year-long study with high school programming learners on three issues: what is the impact of programming on particular mathematical and reasoning abilities? what cognitive skills or abilities best predict programming ability? And what do students actually understand about programming after two years of high school study?

The work of Pea and Kurland has been frequently referenced as innovative because of its focus on the differentiation between *programming language* and *programming environment*, and the importance of developing skills related to both. As they mention in their fundamental study “*On the Cognitive Effects of learning Computer Programming*”.

“*Expert programmers know much more than the facts of programming language semantics and syntax. However, the rich knowledge schemas, strategies, rules, and memory organizations that expert programmers reveal are directly taught only rarely*” [27].

2.2. Research Related to Alternative Programming Methods.

Using an approach similar to that proposed by Pea and Kurland, researchers on Human-Computer Interaction have been focusing on studying the effects of the environment(s) (or the context(s)) during programming tasks. Their results have been reflected in the development of new programming languages, environments, and supporting tools.

One of the most prolific projects related to the creation of different ways of doing programming is the *Natural Programming* project of the Human-Computer Interaction Institute at Carnegie Mellon University. According to Brad Myers, the most renowned researcher on the subject:

“By ‘natural programming’ we are aiming for the language and environment to work the way that non-programmers expect. (...) Conventional programming languages require the programmer to make tremendous transformations from the intended tasks to the code design. (...) We argue that if the computer language were to enable people to express algorithms and data more like their natural expressions, the transformation effort would be reduced” [28].

Many researchers coming from the Natural Programming Project have oriented their research towards studying how learning barriers can develop out of the insecure assumptions that the user-learner do when performing programming tasks on programming systems (or the integration of language plus support software or environment), and how to provide solutions to those barriers.

Two of the most representative works produced by the Natural Programming project are the studies performed by Andrew J. Ko and Thomas D. LaToza, who adapted the initial approach proposed by Pea and Kurland together with design perspectives to propose a new design-oriented approach to the solution of programming learning problems.

In their research, they have observed and interacted with novice and expert designers (mostly software and graphic designers, and content creators) in order to understand how they abstract and represent programming problems, and from there, how they design program’ behaviors and appearance by using different programming languages, programming systems and graphic editing software tools.

There have been many products from those studies: either academic and in form of software tools and programming systems; but there are six studies considered essential for the purposes of this research, namely:

“Designers’ Natural Descriptions of Interactive Behaviors” [3] here, Andrew Ko *et al.* conduct a study with designers and programmers where they described several programming interactive behaviors by using their own language (or way to express, including sketches).

“How Designers Design and Program Interactive Behaviors” [29] in this study Ko *et al.* perform a contextual inquiry where they asked participants (Designers and Developers) to “walk through” their recent projects involving interactivity in order to understand how they perform in different environments and what kind of problems they have.

“How Software Designers Interact with Sketches at the Whiteboard” [23] Where Thomas LaToza *et al.* observe the design activity of 16 professional software designers, coding manually more than 4000 events, in order to analyze the introduction of visual elements in software design processes, and the reasoning activity behind doing sketches to design programs.

“Six Learning Barriers in End-User Programming Systems” [4] where Ko *et al.* perform a study involving 40 non-programmers learning to use the programming system Visual Basic.NET (that consists of a programming language and a programming environment). This study sampled the *insurmountable learning barriers* or the properties or characteristics of this programming system that the students could not understand at all, making them to interrupt their learning process.

“On the importance of Understanding the Strategies that Developers Use” [22] Where LaToza and Myers evaluate current ways of studying the activities and performance of developers and analyze the importance of understanding not only what kind of strategies are developers using to solve programming problems but when are they using them as well.

“Program Comprehension as Fact Finding” [14] Where LaToza *et al.* discuss what is the influence of experience in professional developers work, and how do developers reason or think about the design of a program while doing coding tasks.

3. Proposal of a Programmed Visual Contents Comparison (PVCC) Method to Evaluate Abilities related with a Panoramic Understanding of Programming

Having established that the main interest of this research centers around the identification and assessment of skills related to alternative ways of understanding programming that nowadays programmers, non-developers but professionals in other fields apply and that can lead to more effective solutions or allow the exploration of alternative tools, approaches, techniques to solve programming problems as established in previous sections. In the following pages we present our approach towards a *Programmed Visual Contents Comparison (PVCC)* method to evaluate skills related with the alternative ways they can apply programming techniques recursively based on their knowledge and experience in programming and software tools. As explained in chapter 1 these emergent and opportunistic strategies to do programming allow them to understand programming from a different perspective, this is what we call a *Panoramic Understanding of Programming*.

3.1. What is Panoramic Understanding of Programming?

Rapid changes in the contexts where nowadays non-developer programmers coming from different fields perform their work and in how their final products become widely (and rapidly) used make necessary to know how each field of knowledge can create new ways to approach software design and programming thinking when tackling those changes instead of enforcing standardization (represented in steep learning curves) as a response.

As it was discussed in previous sections, two major issues in early research regarding the way nowadays programmers understand programming are, first, to consider that their way of thinking and doing programming is *ill-behaved*, and second, to assume that they cannot make programs for others or targeted to a wide audience because those products are emergent and lack standardization.

A good number of authors has been thorough in demonstrating incompatibilities between the way these programmers conceive solutions to programming problems and how those problems are solved in a *right way* by professional software developers; but far too little attention has been paid to the identification and evaluation of the different

alternative programming skills that these non-developer programmers coming from different fields apply when conceiving solutions to programming problems, neither to methods to evaluate these alternative skills. [2] [4] [8] [24]

As it was explained in section 1.1.2, these programmers learn and do programming by using not only programming languages, but also software tools that include their own programming language and can be managed through command lines or interface buttons or menus. When they deal with the combination of programming language and interface operation, almost always face learning barriers from both sides. In this regard Ko et al. consider that,

“Visualization tools assume difficulties in imagining the structure of data. (...) learners [do] not face barriers in understanding data itself, but in trying to act on data (such as how to create or modify it)”. [4].

For example: a popular web authoring tool used by Web Content Managers and User Interface Designers to build websites with HTML, CSS and JavaScript among other programming and markup languages is Adobe Muse [30]. This tool has several snippets and pre-made objects that can be selected from an interface, dragged into a visible template of a web page, and will supposedly work at execution time. There are many ways to *act on data* in these software tools: commands and *tricks* to make appear specific pieces of code, or to control diverse processes that usually require a long sequence of mouse clicks, searching around the tool menus or the interface buttons at the same time that the code can be erased, rewritten and changed.

Students learning programming this way 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 found externally; in the end by *sketching* with code [29] [7].

Sketching with code allows individuals not belonging to fields related to software design or programming to adapt their knowledge to be used in programming; in other words, they adapt what they can use of their field-specific knowledge and experience into *designing* a program, therefore placing themselves in a more objective level where they can contemplate what kind of sources they have, what can they pull from those sources and how to combine those sources to achieve what they want; generating then a *panoramic* kind of understanding of programming merged with design concepts [3] [29] [23].

In this sense, a nowadays programmer non-formally educated in computer science or software development who has a *Panoramic Understanding of Programming* combines diverse skills for physical appropriation of objects not related to programming or software design (for instance: drawing, diagramming, getting to know characteristics like: size, color, form, texture) into a programming environment that behaves as a stage where he can explore the relation between objects (things), and actions [7]. The product of this exploration then, is a program or a *sketch*.

This ability of looking at a (programming, or software development) problem in a panoramic way is becoming more and more common among software designers and engineers. Nicolas Mangano *et al* state regarding the ability of *Sketching* as this manifestation of a panoramic or transversal thinking in software design:

“Sketches play an important role in any design process. They serve as an extension of a designer’s own memory (...), help them reason through complex tasks (...), and support them in picturing and evolving hypothetical ideas and abstract concepts (...). Unsurprisingly, sketches play an important role in the software design process as well (...). The design work of software designers at the whiteboard has been examined from a range of perspectives, including idea generation (...), design notations (...), decision making (...), and collaboration.” [23]

3.2. What Programming Skills do we Want to Evaluate?

There is a *thinking shift* in programmers who make *Sketches* when compared with programmers doing formal, development-oriented programming. For a formal programmer the context is different when using a programming language in a text editor, there is little holding up this programmers’ perception of *things* or *objects* in the way explained in previous paragraphs, but instead, other skills become relevant, for example: code reading, syntax mastering, errors detecting, etc.

Meanwhile, programmers doing *sketches* having a *Panoramic Understanding of Programming*, design programs by thinking first on terms of the resources needed to build (the *blocks*), from where can they obtain them, and how to connect them to work, instead of pondering issues like: debugging, optimization or building and handling specific algorithms, that would instead become critical aspects for a trained software developer.

Having the aforementioned aspects in mind, it is our main interest to evaluate the capacity of programmers to do this *thinking shift*, in other words, we want to evaluate the skills of programmers, not only coming from other fields but also software development learners to adapt and combine what they can use of his field-specific knowledge and experience on other software tools and resources when dealing with particular programming techniques.

Ozenc *et al.* refer to the lack of visual handlers in code as the *immateriality of software*, they mention that:

“Most designers explore materials in a studio or workshop where they cut, bend, and play with a material to develop tacit knowledge of what is possible. However, designers cannot easily play with the material of software making the development of tacit knowledge much more difficult”. [7].

Programmers who treat programs as *sketches* bring materiality to code; professionals and students of Web Programming, User Interface Design and other related fields constantly deal with those both *material* and *immaterial* worlds, and learn the fundamentals of both; but since the gap between these two worlds is considerably large, they need to apply their experience and knowledge in their different fields to fill this breach by: using assets and resources from both the graphic software tools or the code sides, identifying patterns inside code that could be repeated, treating snippets and pieces of external code as *blocks* that allow *tweaking* or adjusting, getting to know the internal structure of pre-made objects (e.g. interface components) to look for parts to copy-paste, or searching programming libraries to integrate; [22] these, among other strategies to solve programming problems by using and understanding programming techniques in a panoramic way are in the scope of our interest.

It has been commonly argued that learning how to program brings changes in thought, Pea and Kurland provide a list with seven fundamental changes in thinking abilities that learning to program should bring [27]. Within this list we can find three items closely related with the aforementioned *thinking shift* that nowadays programmers have when understanding programming in a panoramic way. First, as noted by Pea and Kurland, learn to program should bring

“Greater facility with the art of heuristics [or] explicit approaches to problems useful for solving problems in any domain such as planning, finding a related problem, solving the problems by decomposing it into parts, etc.”. [27]

Since a person who has a *Panoramic Understanding of Programming* solves programming problems by putting together a program with (code) parts obtained from different sources, plans a program by handling interfaces, probably synthesizing its design in diagrams [23] or drawing, as we explained, it can be argued that this person applies the aforementioned approaches to problem solving explained by Pea and Kurland to a certain extent.

The authors also mention that learning programming should make a programming student familiar with:

“The general idea that one can invent small procedures as building blocks for gradually constructing solutions to large problems”. [27]

And they also mention that learning programming should provide:

“Generally enhanced ‘self-consciousness and literacy about the process of solving problems’ (due to the practice of discussing the process of the problem solving in programming by means of the language of programming concepts)”. [27]

As explained in the previous section, programmers who have a *panoramic understanding of programming* solve programming problems *Recursively*; or by borrowing the terms used by Pea and Kurland, those individuals are capable of discussing about programming problems not through programming concepts but by bringing his particular knowledge on how to solve similar problems in his field (with its own concepts) into programming.

This alternative way of thinking and doing programming entails a different set of skills that is closely related to those basic or formal programming skills but are apprehended and manifested in a distinctive way particular to each individual, this apprehension of programming concepts, techniques, tools depends on the previous knowledge and experience that this learner as an end-user programmer belonging to a different field possess, as Pea and Kurland mention:

“‘Programming’ is not a unitary skill. Like reading, it is comprised of a large number of abilities that interrelate with the organization of the learner’s knowledge base, memory and processing capacities, repertoire of comprehension strategies, and general problem-solving abilities such as comprehension monitoring, inferencing, and hypothesis generation. (...) Skilled programming, like reading, is complex and context-dependent.”. [27]

3.3. How to Evaluate Skills Related with a Panoramic Understanding of Programming?

Considering that end-user programmers who treat programs as *sketches* solve design problems with programming; when they deal with code in the same way they approach to design, they would most likely lack *affordance*. In other words, because of the aforementioned *immateriality of software*, they cannot receive from a piece of code the similar *talking-back* or feedback than they would expect when, for example, they draw with a pencil on paper; as a consequence of this they have to rely heavily on what they can see (Visualizing) and what they can know about how the program is executing (Guessing) to change it, as mentioned in section 1.2.2.

Ozenc *et.al* argue that designers engage in a conversation with materials when they conceive new ideas, particularly:

“As designers conceive of a new idea or refine the details of an existing idea, the materials they use begin to “talk back” revealing new opportunities and challenges. For example, when sketching with a pencil on paper, designers can explore a product’s physical form, reacting as each line is added to the page”. [7]

Having in mind that a key aspect of having a panoramic understanding of a programming technique is to have *affordance* of it by visualizing its output or result and guessing how it can become if it’s data flow is changed while it is being programmed, we started wondering about the possibility of a nowadays programmer to figure out a programming technique by only visualizing its effects and guessing how is it supposedly behaving.

If this programmer is only shown the *output* of a previously unknown already completed programming sample; could he be able to build the *affordance*, starting from what he sees on screen, to guess based on his particular knowledge and experience in programming what kind of programming technique is modifying the elements of the program?, and besides, is this programmer able to say based in his knowledge and experience with a particular programming technique if two programming samples have or not the same technique used in similar or different ways?, and more important, if this person is capable of building the *affordance*, does this mean that he *understands panoramically* that programming technique?

3.3.1. Why Including Programmed Visual Contents?

As mentioned in section 1.1.2, It is probable that those programmers coming from other fields that manifest having design learning barriers when learning formal programming, instead of having the ability to abstract the problem and conceive an optimal way to put it into working code like formal programmers do, they have the skill of understanding the nature of the problem *visually* first and answer to it by building their own affordance for it.

When facing the creation of new programs, these programmers need to have any sort of previous design: a draft, a script, a (moving) picture to guess what are the resources they would need to build the program and how to configure it. By following this logic, if a programmer is shown the final output of a program he will try to do the same thinking process, therefore guessing what kind of techniques, and tools he would use to build that specific programming sample.

For example, Fig.1 shows the output of a programming sample that uses the programming technique called *Iteration*:

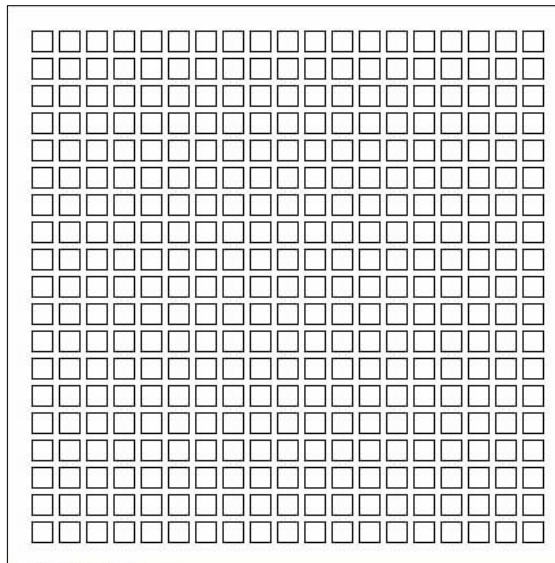


Fig. 1 Output of a programming sample containing the *Iteration* technique

Even without understanding the theoretical particularities or the standard patterns through which the technique *Iteration* can be applied in formal programming; not even knowing the name of the technique, by seeing this picture a programmer whose programming experience is based in software tools may be able to build a program

producing a similar output in short time by *sketching* in any of the software tools or programming platforms he would be able to manage.

In this sense, it is possible to evaluate if an end-user programmer would have a *panoramic understanding* of the technique based on his experience and knowledge of software tools, without knowing about *iteration* itself as a topic of formal programming.

3.3.2. Why Comparing Programmed Visual Contents?

Even though nowadays programmers coming from different fields are capable of building programs by only visualizing or having a visual draft or picture of what should be the final result, and most of the times without knowing the particularities of the programming techniques used as it was mentioned in the previous section, they could have issues at knowing where is a programming technique implemented in different programs; in other words, they may have issues at identifying what is the relevance or importance of a programming technique when used in two different programs, as Myers *et al* reported in their study regarding designers' issues at assigning programmed behavior to their designs:

“Designers do not have a final conception of the behavior before they start, However, whereas iterating on the look of the interface can be easily done by sketching, designers felt it difficult to iterate on the behavior. (...) authoring tools make it difficult (...) to compare two implementations of behaviors side-by-side”. [29]

The most likely cause of this difficulty may be that, as it was discussed in section 1.1.2, since the program design that programmers non-developers who use software tools instead of programming languages perform is in most of the cases emergent and opportunistic, it is carried on at the very moment of programming by *guessing* what kind of function, black-boxed piece of code or tool works to obtain the desired effect; this *guessing* is only possible if there is some way to know what are the effects of specific changes in *real time*; for example: visual programming platforms like the mentioned in section 1.1.2: VVVV [11], EyesWeb [12] and Max MSP [10], all of them have internal tools that allow to see changes in the data flow in real time, without these tools the programmer could not be able to build the affordance needed.

If a programmer like these is shown two output pictures of already finished programming samples and he is said that those two pictures contain the same programming technique only with differences on implementation (i.e. other techniques

are involved) or it's used in distinct parts inside the program, it is almost certain that this programmer would try to start building an affordance of the programs from the pictures and then guess the behavior of each one of them; when doing that he will try to compare his own models of programs to see what is the common technique, where the technique is applied and what is the importance of the use of this technique for each program. The result of this thinking process will depend on his knowledge and experience with the specific programming technique.

For example, Fig. 2 shows a problem where two samples built by using the same code (an iteration process) changing only its parameters are compared; there is no more difficult implementation of the programming technique for neither of them, both samples are similar.

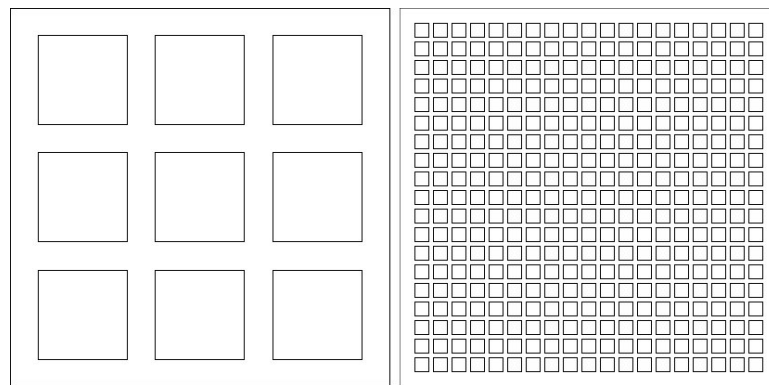


Fig. 2 Sample comparison including the *Iteration* technique

If we ask programmers of different experience and knowledge levels on different software tools which one of the programming samples in Fig. 2 is more difficult to build with programming, we would expect those who have knowledge and experience on how the iteration process is applied on both samples to say that these samples are similar, since they would surely know that both samples are built by using the same technique only changing its parameters.

In the other hand, those programmers choosing one sample over the other as their answer would likely be unaware of the specific programming technique used to build both samples (iteration) and may not be capable of guessing correctly what is their behavior; they would probably give more importance to screen issues (e.g. scale, distance between objects) than to the programming process producing the difference on the output.

This way it is possible to identify which of these programmers could have a more complete understanding of a programming technique which, again, may not be based in

theoretical knowledge nor in formal programming standards but on their own knowledge and experience.

3.4. The Programmed Visual Contents Comparison Method (PVCC).

Seeing that it is possible to evaluate the capacity of an end-user programmer to build the affordance of programs he doesn't know only by looking at its output or final product, therefore being able to guess if a programming technique is being used and to some point know how is it being used, we set out to propose a *Programmed Visual Contents Comparison Method (PVCC)* based on the comparison of two or more displayed images, animations or interactive graphics produced by programming samples; for our purposes we call this comparison a *problem*.

The programmer answering to this *problem* is asked 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 for a problem is defined by the most difficult *programming technique* in both samples; in simple terms, the programmer needs to guess the programming technique from the visual samples to provide the right answer to each problem.

The programmer answering these problems can use any experience and knowledge he could have on programming, regardless of the (software) tools or programming languages he could know or have experienced. The following is an example of the type of problems proposed:

Fig. 3 shows a problem where the sample marked with (1) uses a technique called *Hidden Line Removal* to draw circles that appear to be superimposed, while the program of the sample marked with (2) doesn't use this technique, therefore the correct answer for this problem was decided to be: the sample marked with (1).

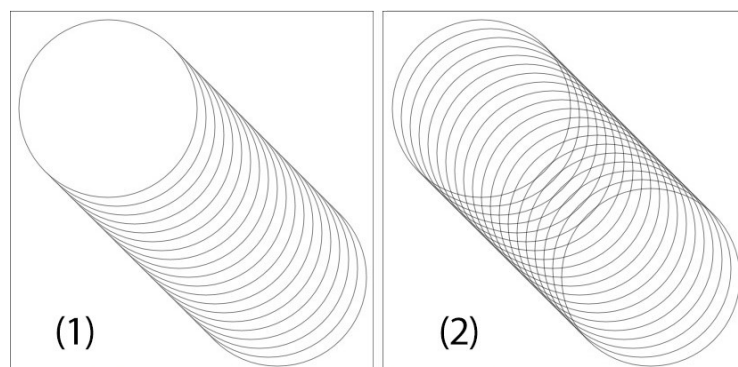


Fig. 3 Problem including the *Hidden Line Removal* technique

Those individuals with programming knowledge enough to know 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 individuals 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 subjects are surely unaware of what kind of algorithm is the *Hidden Line Removal* and how it is applied.

3.4.1. Problems' Degree of Difficulty

Samples for each problem were build considering different programming techniques of different levels belonging to different topics within programming, chosen from different sources like programming books and tutorials [31] [32] [33] [34].

We call *technique* to any programming process or algorithm that can be applied either by directly writing its source code, or also by using a simplified *black-boxed* programming function, or by clicking a button in an interface option or following a series of clicks and cursor movements inside a software tool; that produces a result affecting the final behavior of the product of these actions, that is: a program.

These techniques can belong to basic or advanced levels inside the whole set of techniques or patterns in programming, but we classified those used to build our samples within four categories, namely: basic programming, data processing (reading-writing), data display processes (drawing, coloring, movement) and events (triggered by keyboard or mouse input).

We designed the whole set of problems to have two kinds of difficulty for each one: first, the difficulty of associating images with working programs; to overcome this difficulty, we consider that the subject answering the problems most likely needs to:

- Understand what is each sample doing (how is it moving? what's happening?).
- Identify what elements each program is using to do what it is doing (if there is a movement on the sample, how is it structured 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 is doing).

Second, to identify the most difficult programming process within those composing the program; to overcome this difficulty we think that the student probably needs to:

- Think about, and/or recall from his own knowledge and/or experience:
 - What kind of programming technique can be used to achieve this movement or effect?
 - What is the main effect of each of those techniques? (The subject probably asks himself: if we apply that technique to the elements of each sample, what is the result? And, is that result coherent with some of what is currently happening on the pictures?).
 - How many programming techniques 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 most difficult programming technique for each sample (what is the more relevant programming technique?).
- Compare both main concepts, for both samples.

Following this line of thought, an initial set of sixteen problems where two samples were compared was divided into three types: first, problems having a difficult image-program association, needing more knowledge on images and/or graphic software tools management (Type A); second, problems having an easily identifiable image-program association but needing a deeper knowledge on programming to perform the comparison between programs and the identification of their most difficult process (Type B); and third, problems with both characteristics, where both kinds of knowledge need to be applied (Type C).

3.4.2. Programmed Visual Contents Comparison Testing System

Based on the proposed method we built a web testing system or *PVCC test system* where the first set of problems was displayed. This system was made by using current web standards and problem' contents were developed initially by using the java-based learning-oriented programming language *Processing*, and ported to web by using *Processing.js*.

Processing was selected among other programming languages because, as their creators themselves advertise it,

“Processing is a flexible software sketchbook and a language for learning how to code within the context of visual arts. [underline included by the author of this thesis]”. [35]

We considered an important part of the system to be able to show the code of each programming sample independently so the person who wants to copy, explore and change it could do it by using *processing* directly without having to deal with anything related to the combination of JavaScript and HTML, besides, even when *Processing* has a large amount of simplified functions, it is a learning oriented language so every technique and process done in this language can be applied in similar ways with other programming languages and software tools, at least in a fundamental level. On the other hand, the interface and database were developed using HTML/CSS, JavaScript, MySQL and PHP.

Each problem was built to be straightforwardly answered, having the same main question: *“If you were to make any of the previously displayed samples by using programming, which one do you think is the most difficult?”* and four answer options: *sample 1, sample 2, Both have similar difficulty* and *I don't know*. During the test the person must choose only one answer, then click on a *submit* button to store it on a database and pass to the next problem. Fig. 4 shows an example of how a problem is seen on screen.

The test was thought to be carried on sequentially (one problem after another) and in one try, even when the initial time needed to answer one problem was considered to be 30 sec. to 1 min, this time was extended to 3 minutes, because of later additions to each question like: Data Input to be considered when answering data processing related problems (to be introduced in chapter 5 of this thesis), and questions regarding experience

and knowledge on each one of the techniques asked (to be introduced in Chapter 7 of this thesis). Table I shows the displaying order of the test problems, and the programming concept of each numbered problem.



Fig. 4 Example of a problem on the Web Testing System

Table I: Initial set of problems Including Their Type and Most Difficult Programming Technique

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

In addition, at the end of the test, a report with user's answers per problem compared with their respective correct answers is displayed; and following this, the test subject has the opportunity to answer a brief questionnaire about the whole test experience.

4. Verification of PVCC Method's Potential to Evaluate the Programming Ability of Novice Programmers.

What follows is an account of the initial approach to verify the possibility of evaluating programming ability with the PVCC method. this approach involved performing a test by using the PVCC system with novice programmers belonging to different fields of study.

The analysis of this test's results and feedback were presented in different conferences in Japan [36] [37] and United States [38], and the feedback obtained from these conferences was considered for further enhancements that will be introduced in subsequent chapters of this report.

4.1. Objective and Context of This Chapter

Our objective for this test is to establish if the programming ability of novice programmers coming from different fields, namely: Graphic Design, Game Software Design and IT can be evaluated with the PVCC method.

This test was initially thought to be carried on only with novice programmers belonging to one field having nothing to do with IT and software development together with software developers but, ultimately, we considered appropriate to include novice programmers from careers learning subjects from software development and 3d and graphic design at the same time because the data obtained from their test results could shed some light on what is the difference between a professional from other field who resorts to programming because of a need and a professional from the same field that is at the same time educated as a programmer.

Having this in mind, the test was taken by novice programmers coming from Graphic Design where they apply programming in some cases to create interactive behaviors or animation as a part of a much larger task or work; Game Designers who apply both programming at a development level as well as graphic design, and naturally, novice software developers and programmers.

4.2. Procedures to verify the PVCC Method's Potential to Evaluate Programming Skills of Novice Programmers

As discussed in the previous chapter; it is important to assume that not every novice programmer will have the same experience on the same programming languages or tools, neither that they will be all inexperienced or totally experienced on programming, even though some of these novice programmers have never written or even copy pasted a line of code, as explained earlier, they could have experienced the same techniques by other ways already mentioned. The following are the procedures we carry out to verify the potential of the PVCC method to evaluate programming ability of novice developers from these fields.

4.2.1. Participants' Characteristics and Previous Programming Ability

4 groups of novice programmers currently studying programming in the fields of Graphic Design, Game Development and IT and Software Development were recruited to perform a test based on the PVCC method by using the previously introduced testing system including the proposed initial set of problems.

The programming ability of these groups was reported by their programming professors before performing the test. We gathered them together to share their results at evaluating their student groups' programming performance and to make them know in detail what the experiment consists of and how their report was to be utilized.

The performance reported by these programming teachers (from now on: the report) was compared with the results of our experiment. These results fall under three categories, namely: Problems matching our assumption, Problems mismatching our assumption but having a significant difference, and Problems mismatching our assumption and having a small or no significant difference.

According to the report, the groups differentiated each other by their field of study and curriculum related with programming and/or graphic software tools in the following way:

The first group: Graphic Design (GD) had a curriculum that included lessons where Graphic Software Tools for Photo Edition, Illustration, Desktop Publishing and 3D Modeling were taught together with Web Coding and Web Design. This group studied only programming languages oriented to Web (HTML, CSS, JavaScript, etc.).

The second group: Game Software Development (GS) studied several programming languages such as: C (and its derivatives: C++ and C#) and Java besides of Game Design related subjects such as: Graphic Design Principles, Character Design, 3D Modeling and Animation. Additionally, their curriculum included subjects on Application Programming Interfaces such as DirectX and OpenGL, Web related languages, Algorithm Theory and Mathematics. Graphic Software Tools were used mostly on Game Design classes.

The third group: IT and Software Development (IT) had a curriculum including subjects on programming languages such as: C, Java and Assembler, that were studied together with Algorithm Theory, Web back-end programming and networking. This group did not take lessons about graphic software tools or visual/graphic related programming languages.

This report indicated also a subdivision in levels of knowledge for IT. The group was divided into two sub-groups: IT-1 and IT-2. It also pointed out that IT-2 have received preparation for IT tests such as the JITEE (Japan Information Technology Engineers Examination) therefore, they may be able to demonstrate more knowledge on programming than IT-1.

4.2.2. Assumption Based on Reported Programming Ability

Based on the report, and using the problem types we specified previously for the method on section 3.3.2, our assumption on the results for the performed test is summarized in Table II.

Table II: Assumption on Answers to the Test Problems per Group per Type According to the Programming Ability Reported

	Type A	Type B	Type C
GD	●	×	×
GS	●	●	●
IT-1 and IT-2	×	●	×

Conventions

●	Group with high score
×	Group with low score

According to the report, Type A problems are to be best answered by GD and GS, this could be attributed to GD's knowledge on Images Management and/or Graphic Software tools; IT's knowledge on this area is little to none.

Type B problems are to be best answered by IT and GS, probably due to their deeper foundations on programming; they are capable of managing programming concepts from their base, while GD manages those concepts only through Graphic Tools, possibly having only a technical base on what kind of programming structures are included or used, and lacking training on programming reasoning and deeper conceptual foundations.

Finally, Type C problems are to be best answered by GS, since this is the only group who learn both Programming and Graphic Tools at the same time.

For IT-1 and IT-2, even when both are supposed to answer better the same type of problems (Type B), the report mentions that IT-2 has a higher knowledge in programming than IT-1 so we expected IT-2 to have a better correct answers average than IT-1.

4.2.3. Results and Comparison to Find Significant Problems

The following is a description of how significant problems were found by comparing the difference on correct answers' percentage between groups. The amount of answers per option per problem was 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 III shows the percentage of the total of correct answers and average for each problem per group, highlighting problems with high and low scores.

By using the correct answers percentages, we could establish difference per problems between the four groups by comparing: GD with IT-1 and IT-2; GD with GS, GS with IT-1 and IT-2 and the two IT subgroups.

Having the differences on the correct answers for each group we could see which problems had a significant difference; considering these as representative we performed an F-test of equality of variances and a two tailed T-test to confirm the validity of the difference for each representative problem.

Table III: Percentage of Correct Answers per Problem Highlighting Those with High and Low Scores per Group

	Type A				Type B						Type C						Avg
	#1	#3	#4	#12	#2	#7	#9	#14	#15	#16	#5	#6	#8	#10	#11	#13	
GD	88	72	44	66	31	44	72	63	41	25	88	81	34	19	16	53	52
GS	90	64	31	69	74	54	67	72	44	51	62	85	41	5	26	84	57
IT - 1	63	56	37	39	66	41	66	49	27	41	71	80	37	20	20	63	48
IT - 2	76	56	56	36	80	28	72	68	60	40	64	80	48	20	24	84	56






Conventions  High Score  Low Score

Table IV shows the difference on correct answers' percentage between the groups highlighting the problems having a significant difference for, at least one of the performed comparisons, those are our representative problems.

Table IV: Significant Difference on Percentage of Correct Answers Verified Through T-Test per Group Comparison per Problem, Highlighting Representative Problems

	Type A				Type B						Type C					
	#1	#3	#4	#12	#2	#7	#9	#14	#15	#16	#5	#6	#8	#10	#11	#13
Difference GD vs IT-1	⊙	×	×	⊙	⊙	×	×	×	×	⊙	⊙	×	×	×	×	×
Difference GS vs IT-1	⊙	×	×	⊙	×	×	×	⊙	×	×	×	×	×	⊙	×	⊙
Difference GD vs GS	×	×	×	×	⊙	×	×	×	×	⊙	⊙	×	×	⊙	×	⊙
Difference GD vs IT-2	×	×	×	⊙	⊙	×	×	×	×	×	⊙	×	×	×	×	⊙
Difference GS vs IT-2	×	×	⊙	⊙	×	⊙	×	×	×	×	×	×	×	×	×	×
Difference IT-1 vs IT-2	×	×	×	×	×	×	×	×	⊙	×	×	×	×	×	×	×

Conventions  Questions with Significant Difference on one comparison  Questions with Significant Difference on many comparisons  Questions without Significant Difference

The result for each representative problem belonging to each one of the types previously determined, was compared with our assumption; Table V shows which problems' result matched our assumption and which ones had other outcomes. Each problem has its own particularities regarding measurability and optimization that will be discussed in the following section.

4.3. How Well Can the PVCC Method Evaluate Novices' Programming Ability?

The following part will discuss three types of representative problems (see Table V): those whose results matched our assumption, those not matching our assumption but considered useful to assess programming ability and those having only one significant

difference and whose results mismatched our assumption; additionally, we consider necessary to do some remarks about non-representative problems.

Table V: Representative Problems Matching and Mismatching our Assumption

	Type A			Type B					Type C		
	#1	#4	#12	#2	#7	#14	#15	#16	#5	#10	#13
GD	●	●	●	×	●	●	●	×	●	●	×
GS	●	×	●	●	●	●	●	●	×	×	●
IT-1	×	×	×	●	●	×	×	●	●	●	×
IT-2	×	●	×	●	×	●	●	●	×	●	●

Conventions: ● Group with high score × Group with low score
● × Result mismatching the assumption

4.3.1. Problems whose Results Match the Assumption

As shown on Table V, results for representative problems #1, #2, #12 and #16 matched our assumption, these problems are considered valid to assess programming ability. To this respect, we want to quote problems #2 and #16, used as examples in section 2.1.

Table VI: Nested Iteration (#2) Problem - Answers for Each Group (Percentage)

	Sample 1	Similar	Sample 2
GD	25	31	44
GS	3	74	23
IT-1	12	66	22
IT-2	12	80	8

Table VI shows the percentage of student answers per group to problem #2, which concept is: *Nested Iteration* (see Fig. 2) and belongs to Type B.

As stated in section 2.1, the correct answer for this problem is *the difficulty is similar*, which was selected by a 31% of GD, a 74% of GS, a 66% of IT-1 and an 80% of IT-2. These students found out that, using *Nested Iteration* both the first and the second sample can be performed with the same difficulty; in the other hand the possible lack of programming ability may have influenced the low percentage of GD.

It is worth mentioning that a 44% of GD a 23% of GS and a 22% of IT-1 selected sample #2 as the right answer, while a 25% on GD and a 12% on both IT-1 and IT-2 selected sample #1. These students probably considered the difficulty of both samples based more on screen presentation issues (scale, distance between objects, visual impression) than on how they were programmed. For instance: some GD students could

have thought the second sample was the most difficult because it had more squares than the first one, then involving more steps if performed using a software tool; or, for the first sample's case, some students probably thought that the squares' size or scale needed to be calculated.

Table VII: Hidden Line Removal (#16) Problem – Answers for Each Group (Percentage)

	Sample 1	Similar	Sample 2
GD	25	22	53
GS	51	15	34
IT-1	41	25	34
IT-2	40	24	36

Table VII shows the percentage of answers that the four groups gave to problem #16 which concept is *Hidden Line Removal* (See Fig. 3) and belongs to Type B.

As stated in section 2.1 the correct answer for this problem is Sample #1 and a 25% of GD, a 51% of GS, a 41% of IT-1 and a 40% of IT-2 selected this answer. These students found out that the first sample contained the *Hidden Line Removal* concept. The programming ability of GD almost certainly wasn't enough for the majority to figure out the difference between the two samples, neither to identify the concept.

We may assume that those students who answered *the difficulty is similar*, namely, a 22% of GD, a 15% of GS, a 22% of IT-1 and a 24% of IT-2, probably have a programming ability limited only to simplified programming languages, therefore not familiar with the *Hidden Line Removal* algorithm; they were able to associate both samples only to the simplified functions used to do these samples on those languages. For example: by using Visual Languages like Processing, through a *for* loop and the ellipse and fill functions both samples can be done and modified in their size, color and filling.

4.3.2. Problems whose Results Mismatch the Assumption Useful to Evaluate Programming Ability

By looking at Tables IV and V we can see that problems #5, #10 and #13 have more than one significant differences on the comparisons between groups, but they didn't match our assumption.

The results for problems #5 and #13 were discussed and analyzed together with the group of teachers in charge of the four groups to establish to what extent they could be useful. Since problem #10 was one of the three problems obtaining the lowest score

(see Table III) it is more appropriate to include it in the group of non-representative problems instead.

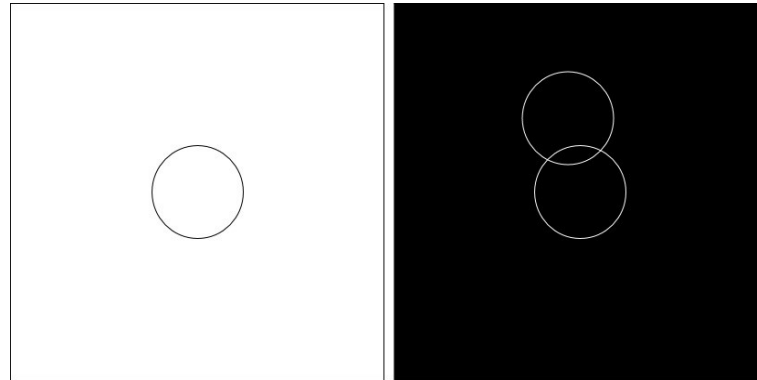


Fig. 5 Appearance of: Boundary Detection (#5) problem

Problem #5, Boundary Detection (See Fig. 5) belongs to Type C. For the first sample, if the mouse pointer hovers over the circle, the background turns black; for the second sample, if the moving circle (replacing the mouse pointer) touches the border of the static circle the background turns black.

Table VIII: Boundary Detection (#5) Problem - Answers for Each Group (Percentage)

	Sample 1	Similar	Sample 2
GD	4	8	88
GS	9	29	62
IT-1	12	17	71
IT-2	12	24	64

We initially considered the correct answer for this problem to be Sample 2 and an 88% of GD, a 62% of GS, a 71% of IT-1 and a 64% of IT-2 guessed our assumption, but in fact, the second sample has other ways to be programmed therefor other ways to be correctly answered. For example: the answer would change to *the difficulty is similar* if we consider that the second sample could contain a bigger invisible circle placed around the centered one, so when the moving circle intersects it, the background turns black just when the borders of the visible circles apparently touch each other.

This issue makes the problem not valid to assess Panoramic Understanding of Programming related with the Boundary detection concept but, looking at the results from students answering *the difficulty is similar*, namely: 8% of GD, 29% of GS, 17% of IT-1 and 24% of IT-2 we can see that a significant percentage of GS and IT-2 students somehow perceived the similarity between the two samples; this aspect led us to think

that they probably went deeper to think about the difficulty of the techniques used on both samples to compare them.

Having this into account, even when this sample resulted inappropriate to evaluate Panoramic Understanding of Programming related with Boundary Detection, we considered it useful to identify student’s potential skills to understand relevant techniques used in a programming sample.

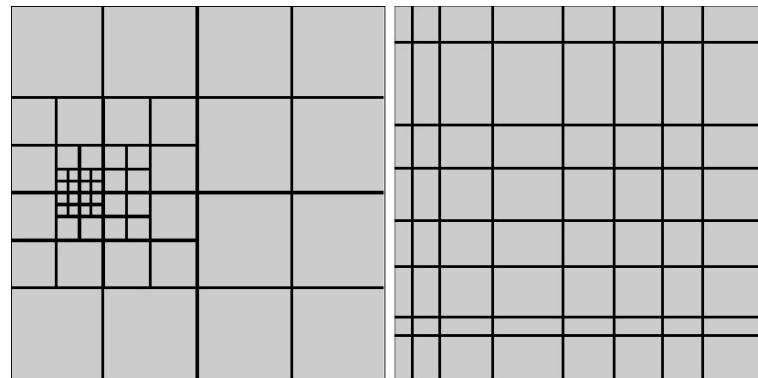


Fig. 6 Appearance of: Recursion and Repetition (#13) problem

Problem #13, Recursion and Repetition (see Fig. 6) belongs to Type C, the first sample changes according to a recursive algorithm when the user clicks over; the second sample draws two crossing lines in the position where the click is performed.

The correct answer for this problem was Sample #1, GS and IT-2 obtained a comparatively high score of 84% both, while IT-1 obtained a 63% and GD had a 53%.

IT-1 and GD had almost the same percentage of people selecting sample #2, and little difference between the percentage of those selecting the difficulty is similar; while for GD this kind of results can be supposed, for IT-1 this could be a sign of their ability level difference with IT-2, some students on IT-1 are probably lacking the ability to identify the common pattern of a recursive algorithm or haven’t studied it yet.

Table IX: Recursion and Repetition (#13) Problem - Answers for Each Group (Percentage)

	Sample 1	Similar	Sample 2
GD	53	16	31
GS	84	5	11
IT-1	63	10	27
IT-2	84	6	10

Furthermore, each time the student clicks on any of the samples, the picture changes; in other words, the number and position of clicks affects the visual impression of each sample, this could have affected the answer too.

4.3.3. Problems whose Results Mismatch the Assumption Useless to Evaluate Programming Ability

Problems #4, #7, #14 and #15 had only one significant difference and their results didn't match our assumption; these problems are not useful to evaluate programming ability.

Within this set, problems #7, #14 and #15 can be affected by clicks or mouse movement; this behavior was programmed together with the code containing the techniques to be evaluated, and in some cases, it was more difficult that those techniques; additionally, these problems were lacking enough instructions or guidance about how to operate them. The following example will provide details regarding this issue.

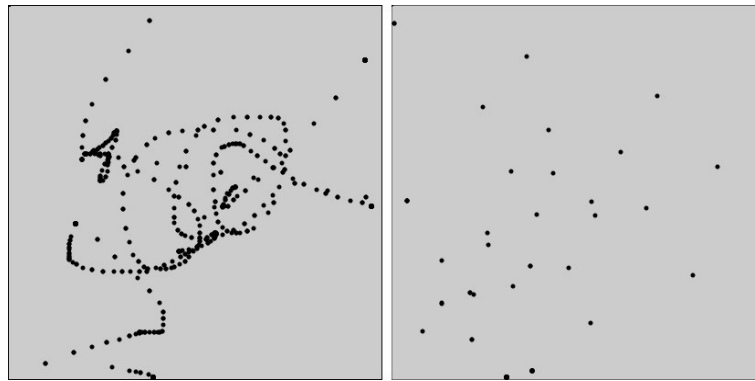


Fig. 7 Appearance of: Timer (#7) problem

Problem #7, Timer (See Fig. 7) belongs to Type B, the first sample draws a point on mouse coordinates every frame; the second sample draws a point each 400 milliseconds only when the mouse almost stops or is really slow.

Table X: Timer (#7) Problem - Answers for Each Group (Percentage)

	Sample 1	Similar	Sample 2
GD	20	36	44
GS	15	31	54
IT-1	39	20	41
IT-2	18	54	28

The correct answer for this problem was Sample #2, and even when the 44% of GD, the 54% of GS and the 41% of IT-1 guessed this answer, but the difference in the percentage of answers between the groups is minimal. Probably the lack of appropriate instructions and the fact that the response of this sample depends on the slow movement of the mouse, made some students think the answer was different.

An additional issue to have into account with this problem is that, even when we implemented the second sample to have a timer (and to be the most difficult), the first one also has a timer: the frame rate, and the mouse makes evident from the moment that it enters the sample's area that this frame rate works as a timer, so that could have been confusing as well.

4.3.4. Non-representative Problems

Table IV indicates that problems #3, #6, #8, #9 and #11 didn't have any significant difference on the comparison of correct answers between the four groups, these problems are not valid to evaluate programming ability. Additionally, as it was mentioned previously, problem #10 belongs to this group too.

For problems #3, #6, #9 the four groups had high percentages of correct answers (see Table III); we think these problems compared samples based on difficult programming processes with samples evidently easier or too basic; in other words, the difficulty level difference of the compared samples for these problems was too obvious.

On the other hand, problems #8, #10 and #11 had low percentages of correct answers (see Table III); these problems were ill-made, most likely because they contained additional concepts on the same level or more difficult than those evaluated, or the programming samples were too similar.

4.4. Conclusion of This Chapter

By performing a test using an initial set of problems based on the PVCC method targeted to novice programming learners and analyzing its results we were able to confirm that the programming skills evaluated with the PVCC method are related to a Panoramic Understanding of Programming.

We were also able to understand that the level and difference on knowledge and experience in programming of the students, the difficulty degree of each problem, and how the programming samples are paired may define how well this method can evaluate programming skills related to a Panoramic Understanding of Programming.

In the same way, we were able to find other kinds of abilities used by the individuals answering the proposed problems; those skills are also related with a Panoramic Understanding of Programming even though they were different to those initially considered.

One source of weakness in this test which could have affected the analysis performed was that most of the programming samples used for many of the problems were ill-made, or misleading, mainly because they involved too many programming techniques with the same relevance or importance, this issue affected the identification of the difficulty for the problems containing them. However, these programming samples may be corrected and paired again to make problems useful to evaluate programming skills related to a Panoramic Understanding of Programming.

5. Extending the Range of Programming Skills Assessable by The PVCC Method

In this chapter we move on to describe our approach to extend the range of programming abilities that can be evaluated with the PVCC method; the proposed approach includes the preparation of new problems with additional criteria for making samples and pairing and modifications to previous samples, problems and testing system.

New problems and conditions for making and pairing new samples were introduced in different conferences in Japan [39] and United States [40], and the feedback received mainly by professors and experts on software development and programming, was used to bring together a new analysis on how to establish what kind of experience and knowledge do developers who answer the test possess (summarized in chapter 7).

5.1. Objective and Context of This Chapter

The main objective for this stage of the research was to expand the range of programming abilities related with a *Panoramic Understanding of Programming* that the PVCC Method can assess.

The most interesting finding that emerged from the analysis of the results of the prototype test described in section 4.3 was that, even though the addition of visual programming and/or visual effects techniques in the majority of the problems was evident, some of the problems included in the prototype test were categorized according to feedback (from professors of the novice programmers) as much more related with programming and data management, placing less emphasis in graphic techniques and more in programming processes.

For example: the problem shown in Fig. 2: *Nested Iteration* used very simple graphic elements (squares), did not require any mouse or keyboard input and did not contain any animated elements. For this problem, novice programmers from IT and Game Development understood that its focus was on identifying what kind of technique was used to manage the data inserted (number of squares) in order to arrange the squares in the way displayed on the samples instead of identifying how the squares were drawn, or display-related techniques like how to determine its size or what happened if the squares were changed to circles etc.

By analyzing the problems targeted towards other programming processes different than visual techniques in the first version of the test we found that it is possible to extend the range of programming skills that the PVCC method can identify by changing and enhancing the samples compared to evaluate more and different aspects of programming.

5.2. New Problems' Characteristics

Having into account the aforementioned possibility of extending the range of assessable programming skills we proposed a new type of problems where 2 or more samples are compared, but, instead of asking to compare only graphical output pictures, both input data (raw text, comma-separated data, associative arrays, XML-like data) and output (either picture or graph) are displayed.

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

Tested subjects answered to two different types of problems: *from which input data sample is more difficult to obtain the displayed output picture?* and *which output sample is more difficult to obtain from the given input data?* depending on how many input and output samples are displayed. A new problem may be composed of: multiple input data samples for a single output picture, a single input data set for multiple output pictures or multiple input data samples for multiple output data pictures.

Results from the prototype test indicated that some students didn't have a clear start point from where to think the problem, therefore having too many ways to go without knowing about what programming techniques were making the difference (for example: which ones required longer time, or more computing resources, or were not optimized).

By including input data, we are attempting to guide the person through what is the intention of the comparison of samples, and we are making sure that the person who knows how to process data in programming, namely, how to read a data file knowing the format, how to store this data and how to apply it in algorithms, or other techniques, will know how the input sample behaves and what kind of processes are added that are different in order to get the output samples from an input data set.

Through the following examples, we explain more in detail how new problems are set up, what kind of programming techniques are involved on each sample and how through the inclusion of input data the intention of the comparison can be identified.

5.2.1. How Input Data Can Guide the Answer: Problems with Same Input data for Multiple Outputs

Figures 8, 9, 10 and 11, all of them belong to one problem and depict the input and output samples of a program to draw Pie Charts. For output samples in Fig. 8 and Fig. 9, population values (percentage) of a number of countries and their regions are displayed in Pie Charts.

In Fig. 8 population percentages are ordered from maximum to minimum and this order is displayed in a clockwise manner, while for Fig. 9 countries are grouped alphabetically by continent and their population percentages are sorted from maximum to minimum, this arrangement is displayed also in a clockwise manner.

If someone would perform both output samples comparison without having the source data used by the program, the output sample in Fig. 9 could be considered as the most difficult to obtain, but this person would surely hesitate about the kind of programming techniques applied to obtain both output samples.

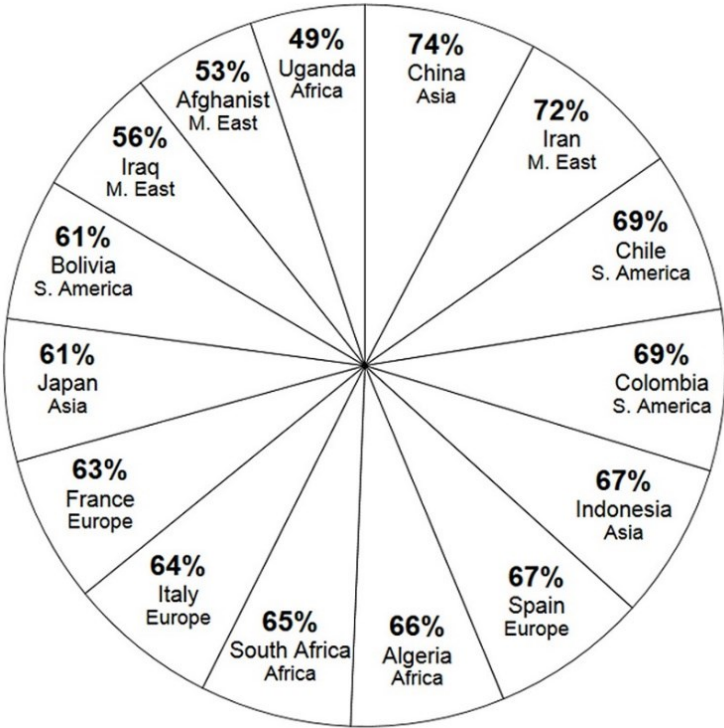


Fig. 8 Pie Chart Program - Output Sample 1

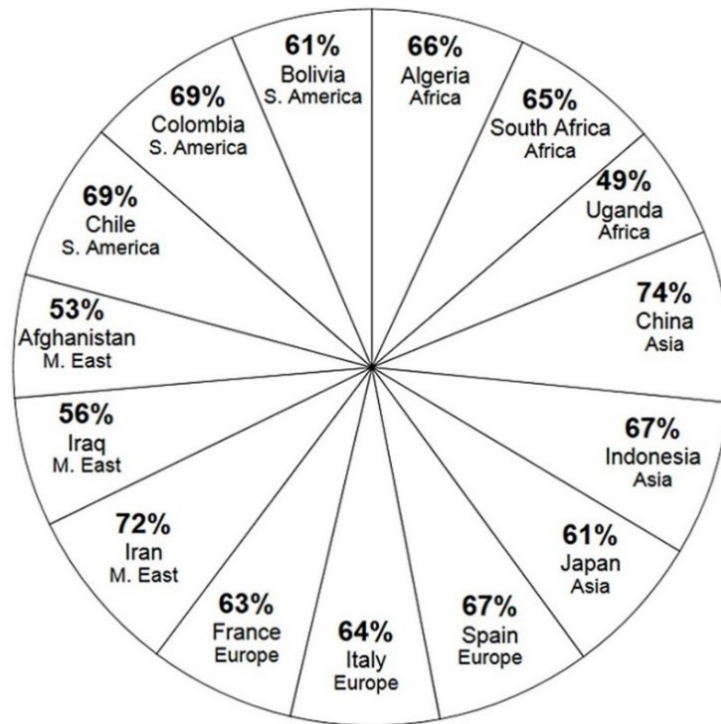


Fig. 9 Pie Chart Program - Output Sample 2

In Fig. 10 the original source input data for both samples is written line by line including country, region and population values; this data is randomly arranged.

Population 15-64 years
Country,Region,Value
South Africa,Africa,65
Chile,South America,69
Iraq,Middle East,56
Indonesia,Asia,67
Spain,Europe,67
Uganda,Africa,49
Algeria,Africa,66
Italy,Europe,64
Iran,Middle East,72
Bolivia,South America,61
France,Europe,63
Colombia,South America,69
Japan,Asia,61
Afghanistan,Middle East,53
China,Asia,74

Fig. 10 Original Input Data for the Pie Chart Program

With the input data in Fig.10 the person performing the comparison can confirm that the output sample in Fig. 9 is the most difficult to obtain, because the program would need to group the countries by continent and then sort the percentage value inside this groups from maximum to minimum to produce this output, while for Fig. 8 the program would only need to sort percentages from maximum to minimum.

But, by changing the source data to the data sample in Fig. 11, where countries are already grouped by continent and percentage values are already sorted from maximum to minimum, the program will process data differently, therefore the correct answer for this problem changes. By applying the data sample in Fig. 11, for the output Sample in Fig. 9 the program would only need to read the data sequentially from the beginning, while for the output sample in Fig. 8 still needs to sort the percentage values from maximum to minimum anyway.

```
Population 15-64 years
Country,Region,Value
Algeria,Africa,66
South Africa,Africa,65
Uganda,Africa,49
China,Asia,74
Indonesia,Asia,67
Japan,Asia,61
Spain,Europe,67
Italy,Europe,64
France,Europe,63
Iran,Middle East,72
Iraq,Middle East,56
Afghanistan,Middle East,53
Chile,South America,69
Colombia,South America,69
Bolivia,South America,61
```

Fig. 11 Change in Input Data for the Pie Chart Program

This way, by changing the way input data is sorted and formatted, the difficulty of obtaining one or another output sample can change drastically; is in this sense that input data is used on each problem to guide the sample comparison evaluation and final judgment.

5.2.2. How Data is Needed to Understand the Program: Problems with Multiple Input Data for a Single Output

The problem displayed in figures 12, 13, and 14 was prepared to ask about data management through the following question: *From which one of the samples in Fig. 12 and Fig. 13 is more difficult to obtain the output picture in Fig. 14?*

Output sample in Fig. 12 shows circles representing Japanese prefectures grouped by region (Kanto and Kansai), and at the same time those regions are grouped in one large group called Japan; the size of each circle represents the area value included in both data samples in Fig. 13 and Fig. 14. Each line of Input data in Fig. 13 contains: Country name, Region name, Prefecture name and Area value. All lines are randomly distributed and data is not sorted.

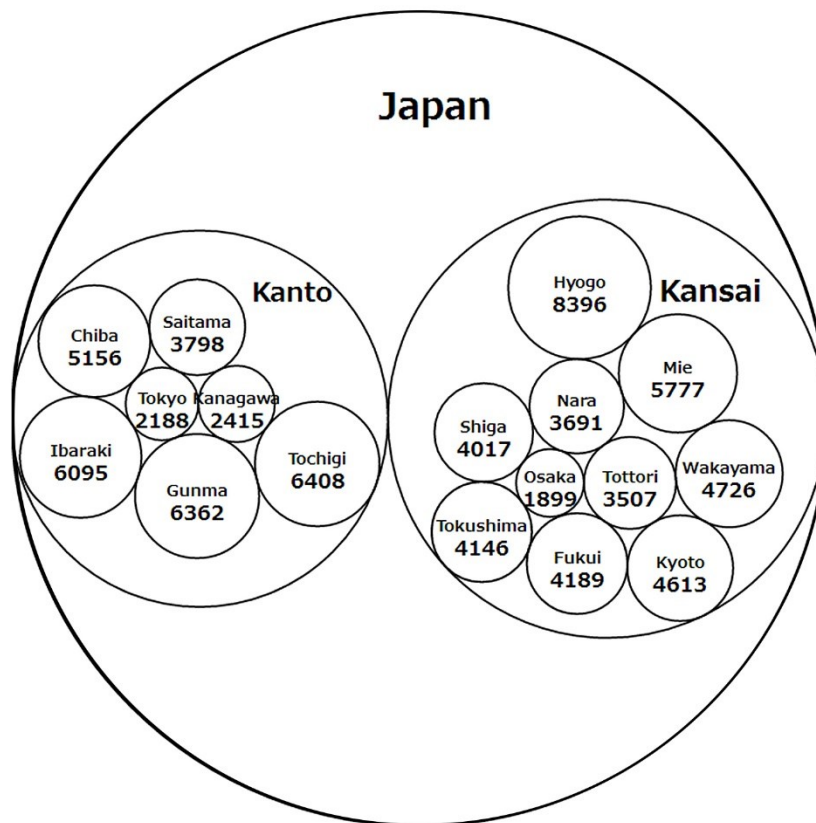


Fig. 12 Circles Arrangement Program - Output Sample


```
Country,Region,Prefecture,Area
Japan,Kanto,Gunma,6362
Japan,Kansai,Mie,5777
Japan,Kanto,Ibaraki,6095
Japan,Kanto,Tokyo,2188
Japan,Kansai,Hyogo,8396
Japan,Kanto,Kanagawa,2415
Japan,Kansai,Fukui,4189
Japan,Kanto,Tochigi,6408
Japan,Kansai,Shiga,4017
Japan,Kansai,Osaka,1899
Japan,Kansai,Nara,3691
Japan,Kanto,Saitama,3798
Japan,Kansai,Wakayama,4726
Japan,Kansai,Kyoto,4613
Japan,Kanto,Chiba,5156
Japan,Kansai,Tottori,3507
Japan,Kansai,Tokushima,4146
```

Fig. 13 Circles Arrangement Program - Input Data Sample 1

As opposed to data sample in Fig. 13, data sample in Fig. 14 is visually organized in a hierarchy; this way of organize textual data is easy to read for a person, but difficult to read for the program drawing the circles.

```
Japan
  Kanto
    Gunma,6362
    Tochigi,6408
    Ibaraki,6095
    Saitama,3798
    Tokyo,2188
    Chiba,5156
    Kanagawa,2415
  Kansai
    Fukui,4189
    Shiga,4017
    Mie,5777
    Osaka,1899
    Nara,3691
    Wakayama,4726
    Kyoto,4613
    Hyogo,8396
    Totori,3507
    Tokushima,4146
```

Fig. 14 Circles Arrangement Program - Input Data Sample 2

In other words, it is more difficult to do a program to get the output picture with the data sample in Fig. 14 because the program will need to perform additional processes to identify which characters indicate the hierarchy (in this case, blank spaces behind each text line); besides, in order to store the area value, the program will need to confirm for each line if it has a parent and children and what is the data of those parent and children, and finally, it will also need to identify the regions and sort the data from the largest area number to the smallest for each region.

It doesn't matter what kind of program is to be included in problems, data indicates which kind of programming processes are more likely to occur in the program receiving it; in the case of the circles arrangement previously discussed, if the input data is not displayed together with the samples, the difference on data reading processes cannot be identified.

5.2.3. How the Absence of Data Can Indicate Difficulty: Problems with Multiple Input Data for Multiple Output

The problem displayed in Figures 15, 16, 17 and 18 was prepared to evaluate knowledge for mapping coordinates in maps, and it asks: *From which program using any of the data samples in Fig. 15 and Fig.16 is more difficult to obtain the output pictures in Fig. 17 and Fig. 18?*

Each line of the input data sample in Fig. 15 includes: name of Japanese prefectures for the Kanto region, area value, prefecture capital city name, and its location coordinate. Input data in Fig. 16 contains border line coordinates for each prefecture map.

The program producing the output sample in Fig. 17 is using only the input data in Fig. 15. It needs to assign each area value as a circles size, then apply a nearest neighbor algorithm to group the circles avoiding circle intersection, additionally it has to calculate the position of the text (centroid of each circle).

The program producing output sample in Fig. 18 is using both input data sets from Fig. 15 and Fig. 16. This program uses the coordinates from Fig. 16 to draw lines that constitutes the border of each prefecture, and the city location coordinates from data sample in Fig. 15 to allocate area and prefecture name texts.

Prefecture,Area,Capital,Capital Location
 Fukui,4189,Fukui,(136.221642,36.065219)
 Shiga,4017,Otsu,(135.868590,35.004531)
 Mie,5777,Tsu,(136.508591,4.730283)
 Osaka,1899,Osaka,(135.519711,34.686316)
 Nara,3691,Nara,(135.832744,34.685333)
 Wakayama,4726,Wakayama,(135.167506,34.226034)
 Kyoto,4613,Kyoto,(135.755608,35.021004)
 Hyogo,8396,Kobe,(135.183025,34.691279)
 Tottori,3507,Tottori,(134.237672,35.503869)
 Tokushima,4146,Tokushima,(134.559303,34.065770)

Fig. 15 Mapping Map and Circles Arrangement Programs - Input Data Sample 1

Prefecture, Border line coordinates
 Fukui,
 (134.375851,35.608377),(134.518312,35.274
 572),(134.405531,35.237561),(134.181714,3
 5.166679),(134.000147,35.349643)
 ... (continues)
 Shiga,
 (134.44359,34.205071),(134.444986,34.2085
 63),(134.647853,34.175741),(134.58605,34.0
 32233),(134.749112,33.832509)
 ... (continues)
 Mie,
 (136.245995,36.290651),(136.342016,36.177
 87),(136.757176,36.085689),(136.825264,35.
 893298),(136.781967,35.795531)
 ... (continues)
 Osaka,
 (136.153465,35.694621),(136.278816,35.661
 101),(136.445369,35.387004)
 ... (continues)

Fig. 16 Mapping Map and Circles Arrangement Programs - Input Data Sample 2

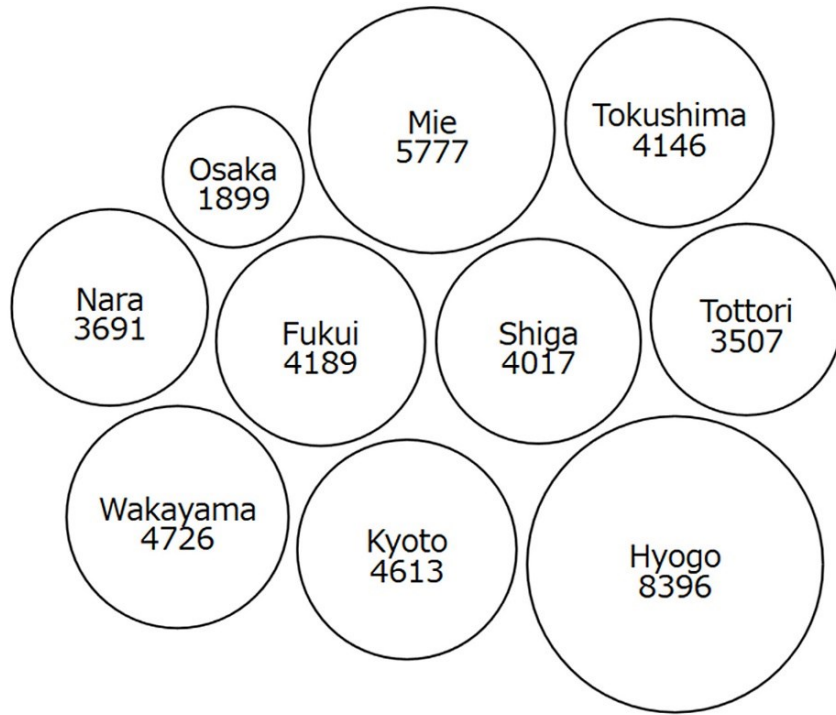


Fig. 17 Mapping Map and Circles Arrangement Programs – Output Sample 1

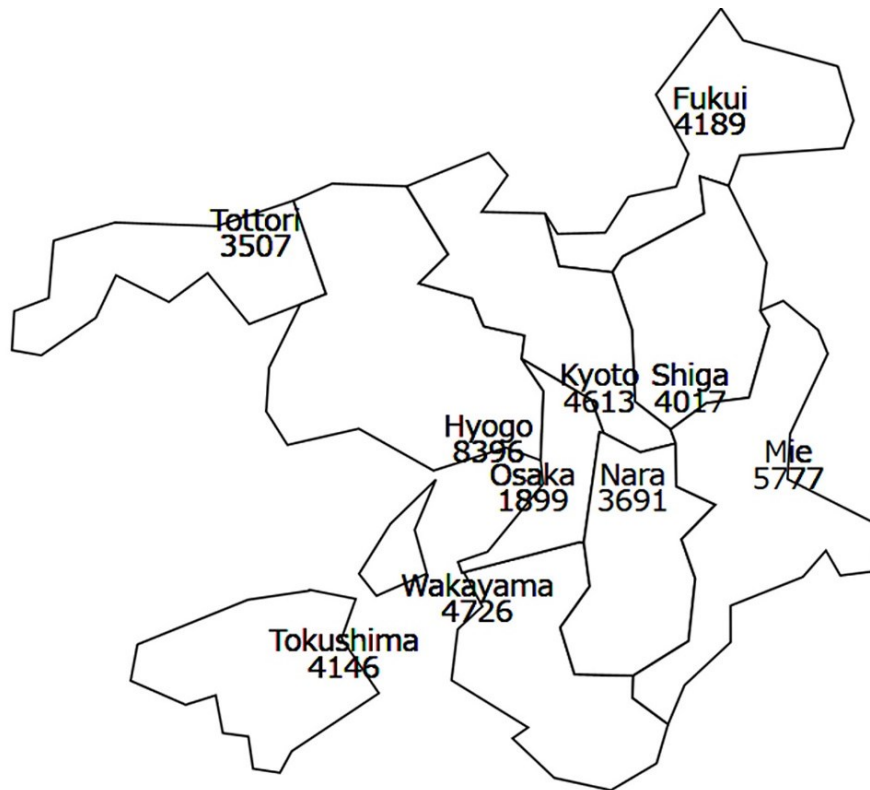


Fig. 18 Mapping Map and Circles Arrangement Programs – Output Sample 2

By including both input data samples the person answering will understand that even when both data samples are used as parameters to draw the map in Fig. 18, the program drawing this map involves simpler processes than the program drawing the circles in Fig. 17; the circles arrangement program applies additional algorithms like: *circles packing* [41] and *nearest neighbor* to group the circles, besides of applying data processing algorithms to sort the map areas and apply them as circle sizes. In this sense the selected output sample should be the output sample in Fig. 17.

Several new problems were prepared following the previous criteria; to see figures and a brief explanation of the most difficult programming process evaluated for each one of them please refer to the Appendix A of this report.

5.3. A Classification for Programming Processes

Following the previously exemplified pattern to set up problems we divide the processes a program needs for any sample in any problem into two categories: Data Processing and Data Display as shown in Table XI.

Data Processing related programming processes can be of three types: Data reading, Data storing and Data Analysis; in Table XI each subcategory for Data processing as well as Data display contains examples of processes that belong to each one, but there could be many more programming processes for each category.

Table XI: Processes Included in Problems Divided by Categories

<p>Data Processing:</p> <p>every programming technique that involves reading a data file and storing and handling its data falls in this category, having three subtypes:</p>	<p>Data Reading</p> <p>for example:</p>	Read specific data formats (tables (e.g. raw text, comma-separated data, associative arrays etc.)
		Execute data Input/Output processes (e.g. I/O functions depending on the programming language)
		String operations or functions for dividing data into manageable strings, identify the most common characters etc.
	<p>Data Storing</p> <p>for example:</p>	Setting up and storing in data structures (e.g. arrays, nested arrays, lists etc.)
		Making references to elements in lists (e.g. pointers)
		Keeping count of the times a process is executed etc.
	<p>Data Analysis</p> <p>for example:</p>	Sort processes (algorithms or functions)
		Different ways to analyze and prepare data (Pattern Recognition, Data mining, complex data structures like Binary trees) etc.
	<p>Data Display:</p> <p>Every programming technique that involves visual or graphic display or arrangement of data falls into this category, for example:</p>	Plotting graphic objects (e.g. Bezier lines, Circles, 3D objects etc)
Scaling and transforming graphic objects (e.g. translate, rotate)		
Allocate an object in a coordinate.		
Assign graphic object properties like color, size etc. from data.		

If we want to identify a programming ability through setting up a problem like the ones previously explained, we can classify this programming ability inside the categorization presented on Table XI and from here it can be decided what kind of processes can be emphasized in each program to be compared.

For example, the problem described in the set of Fig. 12~14 is designed to identify programming abilities related with data management, specifically data reading and data storing. To be precise, the person being able to identify the difference between the two programming samples will surely have the skills to understand how the data format is built (the first data format can be read in an easier way than the second hierarchical data format) how the data is divided into strings (the first data sample will be divided by identifying the comma, while the second data sample needs to analyze each line to identify the strings), and that the program processing the second data sample will need more reading steps to build the hierarchy in lists or structs.

In summary, the classification presented in Table XI is not only for programming processes but it can define programming abilities identifiable with new problems.

5.4. Test Oriented to Professors to Verify the Suitability of New Problems

We wanted to know if any of the new problems were suitable to identify programming abilities related with PUP in the categories mentioned before (namely: Data Processing and Data Display). For this purpose, we built a second version of the test specifically oriented to obtain feedback from programming professors regarding the inclusion of input data, the division in categories and the comparison of the most difficult programming processes.

Programming professors from universities and technical colleges in Japan answered this test in the sequence described as follows:

- 1) Without knowing our assumed answer beforehand, they answered the question: *which output sample is more difficult to obtain from the given input data and/or from which input data sample is more difficult to obtain the displayed output picture*, this depending on how many input or output samples did the problem have.

- 2) After answering the previous question, they read an explanation about the processes to identify per sample and which process is assumed as the one marking the difference for the whole problem; knowing this information they told us if they agree or disagree with our assumption.
- 3) They confirmed if the “new problem” is or is not appropriate to be included in a test to identify PUP related programming abilities.
- 4) Finally, we offered them the opportunity of comment freely about the samples used or the whole problem.

5.5. Are New Problems Suitable for Evaluating Programming Abilities Related to a Panoramic Understanding of Programming?

Table XII and XIII show a summary of how many problems received the respective percentage of matching answers from professors and how many problems professors agreed with the most difficult process to be included; for example, if 6 problems correspond to an 88% in Table XII that means that 6 problems were answered right by the 88% of professors. And if 4 problems correspond to 100% in Table XIII that means that 6 problems did receive an agreement by professors regarding the most difficult process to be evaluated. (Note that percentages don't correspond to 100% because they are rounded up).

Without knowing the assumed answer beforehand, more than 75% of the total of professors' answers matched the assumption regarding the programming processes through which the difference on the comparison for each problem can be determined.

Within the 25% of not matching answers, more than 70% of those incorrect answers were: *the difficulty is similar* this could indicate that, even when the data is present, it is incomplete or is not working as expected to establish the difference between the samples.

Also, within the 25% of incorrect answers, more than 60% of professors after reading the explanation considered the problem appropriate to identify programming abilities, these aspect is likely to be related with a lack of concision in data or graphic

samples; probably these elements for themselves aren't enough to guide a person towards identifying the required difference between programs.

Table XII: Amount of Problems Receiving Correct Answers (High Percentage Range)

percentage of correct answers received	amount of problems
100%	1
88%	6
75%	1
63%	2
50%	0

Table XIII: Amount of Problems where Professors Agreed with the Most Difficult Process (High Percentage Range)

percentage of "agree" answers received	amount of problems
100%	4
88%	2
75%	1
63%	1
50%	0

Table XIV shows how many problems were considered “appropriate” by a high percentage of professors and Table XV shows how many problems were considered as “needing fix” by professors; for example, if for Table XIV 2 problems correspond to an 88% that means that 88% of the professors considered 2 problems as appropriate, and if in Table XV 4 problems correspond to a 13% that means that 4 problems were considered as needing fix by the 13% of the professors. (Note that percentages don't correspond to 100% because they are rounded up).

Table XIV: Amount of Problems Considered Appropriate by A High Percentage of Professors

percentage of "appropriate" answers	amount of problems
100%	2
88%	2
75%	3
63%	0
50%	1

Table XV: Amount of Problems Professors Considered as Needing Fix

percentage of questions considered as "Needing fix"	amount of problems
0%	3
13%	4
25%	1
38%	2
50%	3
63%	1

5.6. Conclusion of This Chapter

Even though the emphasis on visual programming and/or visual effects techniques in the majority of the problems of the prototype test was evident, through the addition of input data to comparisons understanding how to use it to guide the answer to a problem, how it is needed to understand the behavior of programs producing the output pictures, and how the lack of this item can indicate difficulty, and also by simplifying the programs and sample pictures we were able to build a new type of problems based on the PVCC method.

By performing a new test with professors of programming and experts having a set of new problems we were also able to verify that some of the new problems were suitable to evaluate skills related with a Panoramic Understanding of Programming.

For problems that were considered as not suitable to evaluate programming abilities related to a Panoramic Understanding of Programming not only of the second test with new problems but also for misleading or ill-made problems from the first test, professors and experts identified mistakes and points of confusion and suggested optimal ways to correct them, the following section summarizes the main comments and recommendations and presents proposed fixes.

6. How to Create Suitable Problems for the PVCC method?

This chapter discusses how to build appropriate problems to be included in any test that applies the PVCC method according to the Feedback provided by professors and instructors of the four groups of novice programmers that participated in the first test and professors who took the second test to verify the suitability of new questions (from now on the *professors*).

Professors recommended to adjust and change certain aspects of many of the problems after experiencing both tests and analyzing their results. Their recommendations are summarized in 3 topics: that we consider fundamental when building new problems based on the PVCC method: Identification of the most difficult programming technique in a problem, balance of complexity and concision of problems, and revision of the Programming techniques used for each problem.

6.1. Identification of the Most Difficult Programming Technique

The main aspect that affected the results for all groups in the previous experiment is the identification of the most difficult programming technique used in each sample. Some novice programmers weren't able to identify the most difficult technique for most of the problems or erroneously considered the wrong kind of technique; this lead us to think that it is not clear enough what is the most relevant technique in each problem.

As mentioned in section 3.3.1, samples were selected and paired considering two difficulties: the difficulty of associating images with the compared programs and the difficulty of identifying the main, most difficult process for both samples on each problem but, seemingly, in some problems there were additional processes that were equally difficult and/or were more difficult that the ones we considered initially. However, for representative problems in the experiment for this stage, some of the novice programmers in the first test were able to perceive the desired process and answered correctly.

In this sense, professors thought that it is necessary to establish the difference of difficulty between having to deal with the specific code or algorithms for the most difficult techniques and deal with the other aspects of the programs for each problem. There may be difficult techniques in each problem that could be relevant, but the

programming techniques used to make the output pictures could be as difficult (long, tedious, hard) as writing the algorithms we think are the most difficult.

The aforementioned misleading when identifying which is the relevant difficulty on each problem's sample and in the sample comparison as a whole can be perceived in some of the problems included on the first test carried on with novice programmers, particularly one of the problems whose results mismatch the assumption useless to evaluate programming ability explained in section 4.3.3 which serves as an example of problems that can be corrected by identifying more clearly what is the difference on difficulty of each sample:

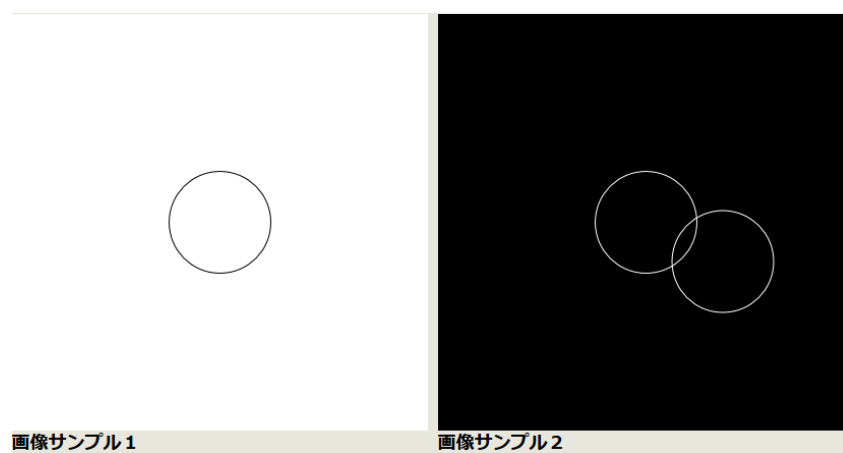


Fig. 19 Appearance of the problem used on the initial test to evaluate the technique: Boundary Detection

Fig. 19 shows a problem used on the initial test to evaluate the technique: *Boundary Detection*; for the first sample of this problem, if the mouse pointer hovers over the circle the background turns black, while for the second sample if the moving circle (replacing the mouse pointer) apparently touches the border of the static circle the background turns black.

By following this logic, we initially considered the correct answer for this problem to be *Sample 2*, but in fact the second sample has other ways to be interpreted. For example: the correct answer would change to *the difficulty is similar* if we consider that the second sample could contain a bigger invisible circle placed around the centered one, so when the moving circle intersects it, the background turns black just when the borders of the visible circles apparently touch each other.

But again, the correct answer would be *sample 2* if we consider that the second sample uses a specific programming technique to replace the mouse pointer with a circle, that is not used in the first sample.

There are, then, several programming techniques with different difficulties included in this problem, and to correct this sample and make it useful to evaluate PUP, it is necessary to add complementary resources to guide the test participant towards focusing on the most difficult technique.

6.2. Balance of Complexity and Concision of Problems

Another issue highlighted by the professors is that some of the problems in this test included data and graphs very difficult to read or interpret. There are comparisons where the difference cannot be identified, not even by reading the explanation provided for the most relevant programming technique of each problem at the end of the first test (individual results report) neither when interpreting the input data of each problem of the second test. In the other hand, some comparisons are too obvious, or the difficulty difference can be easily identified by anyone, doesn't matter if the person taking the test has experience on programming or not.

This can be solved by making the problems more readable, succinct, and easy to follow. In other words, we need to make simpler samples and include other resources to help the test participant understand how to operate them, explaining what the samples are showing and suggesting as well important points to have into account while operating them.

The aforementioned difficulty at reading or interpreting PVCC method based problems can be exemplified with some of the non-representative cases of the test carried out with novice programmers. As explained in section 4.3.4, three problems which obtained the lowest percentage of right answers were ill-made, most likely because contained additional concepts on the same level or more difficult than those evaluated, and/or the programming samples were too similar, so their main purpose was too difficult to identify; we will take one of the aforementioned problems as an example of an ill-made or misleading problem, that can be enhanced by explaining how does it work or what is the main point to start thinking about its relevant difficulty.

Fig. 20 shows a problem used on the initial test to evaluate the technique: *Change according to time*; for the first sample of this problem, each second a line with random

starting and ending points appear, while for the second sample a horizontal line apparently moves upwards during the same second.

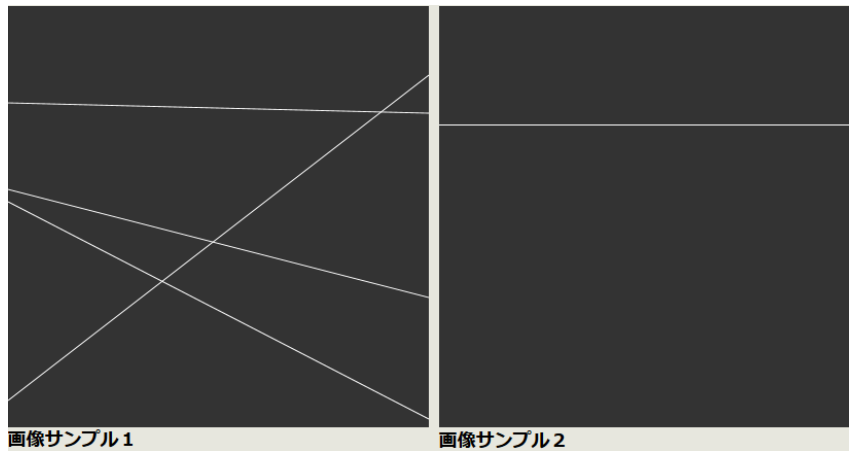


Fig. 20 Appearance of the problem used on the initial test to evaluate the technique: Change according to time

The correct answer initially considered for this problem was *Sample 1* since we considered the random change of the initial and final points of the lines appearing each second more difficult than the change of the line in the second sample, but, if we would consider the programming techniques applied in the second sample additional to the time management technique, namely: a line erasing and rewriting technique, together with a programming process to reset the position of the next line to be written when the previous written line reaches a point beyond the upper extreme of this sample, the amount of programming processes with almost the same level of difficulty incorporated in both samples becomes greater and, in addition, the *change through time* initially intended to be the technique to evaluate results being similar for both samples, thus making the difference of difficulty of the other techniques involved much more relevant, consequently making the identification of the initial technique too difficult.

This problem can become useful to evaluate PUP related programming ability regarding the programming technique: *random* if the amount of techniques involved into each one of the samples is reduced; for example, the erasing and rewriting technique from the second sample can be changed to only a writing process similar to that of the first sample without the randomness of the lines' initial and final points.

This problem can also be enhanced by adding guidance resources that help to make clear where to start thinking the relevant programming technique (without exposing the answer).

6.3. Revision of Programming Topics and Samples

A third aspect that needs to be thoroughly revised according to the suggestions provided by the professors is the set of programming topics or subjects underpinning the samples used on the problems mentioned in sections 3.3.2 and 3.3.3; the initial thought was to have as a reference a somewhat wide range of programming techniques taught in basic programming curriculums. However, the professors noted that our references and sources from where we compiled these techniques were too focused on visual programming subjects, obviating several other important topics that are contained into curriculums for IT and Software Programming courses. As a consequence of this the majority of the test problems are predominantly related to visual programming techniques.

Correct answers for all problems were selected having into account the difficulty of the aforementioned programming subjects, in that sense we considered only one correct answer, thus omitting the possibility of each one of these problems to be variously answered by people with different knowledge levels and fields, having different ideas of programming techniques 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 needs to be revised, having into account actual and more general 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 problems whose results matched our aforementioned assumption are a reference of the necessary level to answer a specific problem on a specific programming subject for a specific type of people; having this in mind, we can consider to follow the same pattern these problems have to build and test samples not related with visual programming.

7. Verifying PVCC Method's Potential to Evaluate the Programming Ability of IT Experts in Relation to Programming Experience and Knowledge.

This section describes our approach to verify if the PVCC method, including previously proposed new problems and changes suggested by professors, can evaluate the programming Abilities related with a Panoramic Understanding of Programming of expert developers and programmers in relation to their own programming experience and knowledge.

In this opportunity, we introduce complements to the PVCC method oriented to ask tested subjects about their experience and knowledge of programming and software tools, not only tools that support programming activities but all kinds of software tools.

7.1. Objective of This Chapter

For the third stage of this research we wanted to know if by applying the PVCC method, we can evaluate the programming ability of IT (Information Technology) experts in relation to their own experience and knowledge at dealing with specific programming techniques while using software tools and programming languages.

As discussed in chapter 2, due to recent changes in the context of many non-computing related fields, most of the software tools used in these fields include or comprise programming options to achieve effects, calculations, processes etc.

As a consequence of this, end-user programmers who constantly use and learn these tools together with complementary programming languages deal with a *material* and an *immaterial* world, since they have to manage *objects* that they can see and manipulate, in addition to *code* that in most of the cases is hidden and difficult to understand and manipulate.

End-user programmers learn the fundamentals of both worlds; but since the gap between these two worlds is considerably large, they need to apply their experience and knowledge in their different fields to fill this breach, using assets and resources from the visual and the code sides: possibly identifying patterns inside code that could be repeated or getting to know the internal structure of pre-made objects (e.g. interface components) to look for parts to copy-paste, or searching programming libraries to integrate inside these tools, among other actions. [22]

The aforementioned usage and learning of programming techniques that mix both visual objects and code in different ways makes the programming knowledge and experience of each end-user programmer different; and it is in the scope of our interest to know how different is this knowledge and experience, and how this difference affects the way an end-user programmer understands *panoramicly* each programming technique he learns or uses.

7.2. Procedures to verify PVCC Method's Potential to Evaluate IT Expert's Programming Abilities in Relation to Programming Knowledge and Experience.

As explained earlier, our intention with the new test based on the PVCC method is to know what is the experience and knowledge of each participant IT expert on each programming technique asked, and to compare this information with the results of the test to determine, first, if the problems are sufficiently good to evaluate experts' programming ability, and second, what kind of programming knowledge and experience do programmers need to possess to answer a PVCC method-based problem correctly.

For carrying on this new test, we received the cooperation of the staff members of a Japanese IT company with different backgrounds and accumulated experience.

We must assume that not every participant IT expert will have the same experience on the same programming languages or tools, neither that they will be totally experienced on programming, even though some of these experts couldn't have any experience on data display programming techniques, as explained earlier, they could have experienced them by other ways already mentioned (i.e. by modifying code containing those techniques).

The following are the procedures we carry out to verify the potential of the PVCC method to evaluate programming ability of expert programmers.

7.2.1. Consolidation of Modified Problems.

As we mentioned in chapter 6, our tests included mostly visual programming problems and we only considered one kind of correct answer for each problem; as a consequence, probably those programmers who had not managed visual programming techniques had issues at detecting what was the difference between the relevant or most difficult programming techniques we asked for and the rest of the program processes (according to the categories presented in table XI), but on the other hand, those

programmers who didn't have issues and could differentiate the relevant programming techniques from the other elements of each sample, at least in one problem, could have developed a Panoramic Understanding at least of that technique.

With the test performed during the initial stage of this research we evidenced that, those programming abilities related to PUP demonstrated by novice programmers diverge according to each person's knowledge and experience of separate programming techniques; we understood that each problem evaluates a single technique that can be or not mastered by programmers and end-user programmers as well since as discussed in chapter 1, even when they learn about them in different ways, both professional developers and end-user programmers manage patterns or schemas to perform different programming processes, those schemas change according to the knowledge they possess and the experience they have accumulated and, in the case of end-user programmers, the experience and knowledge in their respective field should be considered together with the experience in programming [14].

Considering the aforementioned aspect, and in order to consolidate a new test with problems based on the PVCC method that complies with the participant company requirements, first we meet with representatives of the cooperating company to discuss about the content of the test, specifically what problems of the finished set of twenty problems did they consider could be included into the test oriented to their employees.

Company representatives suggested to include the twenty problems but besides, they proposed a classification for the whole set of problems since the time their staff had to dedicate to this test was short; they suggested to divide the test into a set of ten main mandatory problems, and ten secondary optional problems.

Following this requirement, we proceed to select both sets of problems; the main set was selected based on the best performing problems from the previous test with novice programmers and the test with professors and experts, the optional set was selected among those who performed well but didn't have the best results. For this version of the test we also provided the individual with the option to skip questions.

We thoroughly considered also the classification for problems provided in section 5.3; in this section we mentioned that, to identify a programming ability with a problem based on the PVCC method, this programming ability can be classified as shown in the aforementioned section and from here it can be decided what kind of processes can be emphasized in each program to be compared. Table XVI shows the final organization for

the main and secondary sets of problems to be displayed inside the test including its evaluated technique and category.

Table XVI: Sets of Main and Secondary Problems Composing the New Test for Expert Programmers, including Most Difficult Technique and Category

Main Problems		
Number	Most Difficult Programming Technique	Category
1	Programming Loop	Data processing
2	Previous Position Storing	Data processing
3	Picture pixel array management	Data display
4	Recursion	Data processing
5	Empty Area Detection	Data display
6	Hidden Line Removal	Data display
7	Distribute Objects According To Data	Data display
8	Multiple Data Sources	Data processing
9	Visualization According to Data Analysis	Data display
10	Data Format Reading	Data processing
Secondary (Optional) Problems		
11	Erase and Rewrite	Data processing
12	Invisible Events	Data display
13	Time, Speed and Distance	Data display
14	Clickable Area	Data display
15	Random	Data processing
16	Trigonometric Animation	Data display
17	Lists Elements Inserting	Data processing
18	Mapping Data into Shapes	Data display
19	Data Hierarchy	Data processing
20	Mapping Data: Scaling and Measuring	Data display

The characteristics of every problem selected for the new test is described in greater detail in Appendix A; this appendix also presents the new structure of the test, the way the problems and the questionnaire were shown and what were the categories for each question of the questionnaire that, will be introduced in the following section.

7.2.2. Questionnaire and Guidance on Experience and Knowledge of Programming and Software Tools.

Section 6 addressed a number of important suggestions provided by professors and experts participating in previous tests; this section showed how by adding complementary guiding resources and correcting according to professors' suggestions, poorly constructed and/or misleading problems can become useful to evaluate programming abilities related with PUP.

In addition to what has been already stated in the aforementioned section, professors and experts suggested also that if we wanted to establish a difficulty difference between problems, it would become necessary to know what is the experience and knowledge in programming and use of programming techniques of each person answering each individual problem; for example: if they have written code from scratch:

what techniques are they familiar with?, of if they are also capable of identifying those basic techniques inside more complex algorithms or complete programs; and, if they have used software authoring tools it could be interesting to know if they have copy-pasted code to do a program, or if they have used pre built objects, or libraries, among other methods already mentioned. They could have also used similar techniques not in programming but as options in software tools; as it was already indicated, most of the software tools currently have options or include commands to make use of programming techniques automatically, without having to deal with code.

To obtain this information regarding each participant we built a questionnaire about experience and knowledge in programming and software tools to be included to each problem of the test based on the PVCC method, this questionnaire consisted of two main parts:

The first main part is the *guidance*: there were three types of explanation texts for each problem: the first type was a brief *hint* text presented at the first time the problem was displayed, this hint lead towards the most appropriate point of view from where the person answering can start analyzing the problem, and it was not related with the problem's answer.

The second type was a text that indicated *how to handle* each sample; this text was presented at the same time the sample comparison was displayed and it was necessary to help solving one of the main difficulties reported by people on the feedback for both of the performed tests, namely, the difficulty or lack of knowledge on how to operate and interpret some samples that resulted in mislead answers.

For problems involving programming samples with animations and keyboard or mouse input, this text explained concisely how to operate each sample to obtain the desired effect or result, for example: the text explained how to move and position the mouse inside each sample if there is a need to do it, or how to restart an animation that became full of drawing objects therefore confusing, etc.

For problems displaying data input and graphic output at the same time, this text explained what each data sample consists of and what is the graphic output related with that data input.

The third type was a proper *explanation* that clarified what kind of programming technique we were evaluating and why was that technique the most difficult one for each problem; this text was presented after the person answered the sample comparison, and

served as a reference point for the person answering the test to consider his own knowledge on the technique in question in order to be able to answer the next part of the questionnaire.

The second main part was the *survey*: it consisted of a series of questions about the evaluated programming technique in regard to two topics: first: the understanding of the *explanation* text provided previously, and second: the possible knowledge and experience the person taking the test could have had with this particular technique in software tools (maybe using it by inserting commands or pushing buttons on an interface, or probably by handling other functions of software tools related to this technique) or programming (not only languages but also libraries that could apply this technique, or code snippets that contain it; it also can be found inside more complex code).

These questions were classified into three main types:

- 1) *Awareness or perceptiveness of the main programming techniques*: for these questions the person answering the problem clarifies his understanding on the explanation provided and acknowledges if he or she would have realized about the use of the process without reading the explanation text.
- 2) *Experience on handling similar techniques on software tools*: for these questions the person describes his previous experience with similar techniques on software tools (not programming languages), for each problem we provided an example of what kind of software tools could include similar techniques.
- 3) *Experience dealing with the programming technique*: in question not only in programming languages but by using programming libraries or code pieces etc.

Type 1 questions were yes/no questions; questions type 2 and 3 were asked by using the formula: *have you ever...* (e.g. *have you ever used loops in programming?*); to answer these questions a Likert scale was used, the answers for these scales were as follows:

- **Never**: as in: *I have never used loops in programming.*
- **Once or Twice**: as in: *I have used loops in programming once or twice.*

- **Many times:** as in: *I have used loops in programming many times.*
- **I'm used to do it:** as in: *I'm used to do loops in programming.*

As mentioned, Appendix A contains the complete list of problems with its respective list of questions. From the list presented in this appendix no more than two questions of type 2 and two of type 3 were selected for each problem.

7.2.3. Participants Characteristics Including Previous Experience and Knowledge.

A group of 9 IT experts, members of the participating company's staff were recruited to perform a test with the proposed new problems and answer the questionnaire per problem in the way described in previous sections.

Industry experience and self-reported expertise data were collected with a short demographic survey completed before starting answering the test. They were asked to list the programming languages they have learned during their life as students or as professionals. Table XVII shows the collected data.

Table XVII: Participants Self-Reported General Working Experience, Specific Programming Experience, and Knowledge on Programming Languages

Participant	Type of Experience	General experience (years)	Programming Experience (years)	Languages Knowledge
A	Administration	Less than 1	1 to 3	HTML, PHP, JavaScript
B	System Operation	Less than 1	No experience	No experience
C	Development	More than 10	More than 10	Visual Basic, VB.net, Java, C#
D	System Operation	5 to 10	1 to 3	No reported
E	System Operation	more than 10	1 to 3	Cobol, Visual Basic, C
F	Contents Development	Less than 1	1 to 3	C, Java, JavaScript, Basic
G	Infrastructure	1 to 3	1 to 3	C, Java
H	Development	More than 10	5 to 10	Java, PL/SQL, VB.net, AccessVBA
I	System Operation	More than 10	No experience	No experience

We chose a small sample because of the expected difficulty and time spent on answering the questionnaire and furthermore because we expect to continue doing similar tests with such reduced groups of experts at different companies as a part of the future work of this research; this in order to keep verifying if the problems based on the PVCC method can evaluate the programming ability of programmers with a different profile and different background (different universities and schools, different workplaces).

Excluding three participants who, despite of their experience in system operation and maintenance did not report having learned any programming language, all participants reported having experience with a wide range of languages such as: COBOL, Visual Basic, C, C#, Java, JavaScript, PHP etc.

Two of the three participants experienced in System Operation and Maintenance reported not having any programming experience, and one of them had less than a year of professional experience. Within the remaining 7 participants, 5 of them had more than a year of programming experience, and two of them more than 5 years of experience.

It was fundamental for us to have people experienced in an IT field not closely related with software development like systems design and administration because the data collected from these subjects could give us closer insights to one of the main objectives of this test that is to find if the experience of people expert in fields not directly related to programming or not doing programming as a common job is capable of answering correctly some or all problems based on the PVCC method.

7.2.4. Assumption Based on Reported Experience and Knowledge

As mentioned in section 7.2.1, two sets of problems were set up for this version of the test; our assumption is based on the main set of ten problems since we were not able to assure that all participants answered all problems from the optional set.

Based on the information provided by the participants, and using the problem order and types already specified, our assumption on the results for this test is summarized in Table XVIII. According to the experience and knowledge initially reported, we expected most of the participants with enough experience in formal programming languages (not specific for visual programming) to answer correctly those problems related with data processing *but* having also a good probability of answering correctly those problems related with data display.

On the other hand, we considered that those not experienced in programming or only experienced in software tools have a high probability of answering correctly the problems evaluating the easiest techniques that may be used in software tools commonly, while the most difficult techniques requiring a deeper programming knowledge may be unknown by them.

Table XVIII: Assumption on Answers to the Test Problems per Type According to the Programming Ability Reported

#	Evaluated technique	Category	Experienced in Programming	Experienced in Software Tools	Not experienced
1	Programming Loop	Data processing	●	●	×
2	Previous Position Storing	Data processing	●	●	×
3	Picture pixel array management	Data display	△	△	×
4	Recursion	Data processing	●	△	×
5	Empty Area Detection	Data display	△	×	×
6	Hidden Line Removal	Data display	△	×	×
7	Distribute Objects According To Data	Data display	△	×	×
8	Multiple Data Sources	Data processing	●	×	×
9	Visualization According to Data Analysis	Data display	△	×	×
10	Data Format Reading	Data processing	●	×	×

Conventions:

●
△

 Right

×

 Wrong

7.3. Is It Possible to Evaluate IT Experts' Programming Abilities in Relation to Their Experience and Knowledge by Using the PVCC Method?

What follows is an account of how the test results relate with our initial assumption and how the correlation between the reported experience and knowledge and the test score was found and validated. Table XIX shows the correct answers per participant and correct answers per problem including the average of both.

As already explained, the objective of this test was to establish the relation between the personal experience and knowledge regarding the application of the programming technique evaluated either in programming and software tools with the answer provided to the sample comparison.

Of the nine participants, two developers who reported having more programming experience (participants C and H) had the highest score on the test: 9 and 7 correct answers respectively; in addition, two participants whose background is in Infrastructure and System Operation and Maintenance, who reported having 1 to 3 years of experience

in programming and knowing languages like Java, C and Cobol answered correctly to 6 and 7 problems respectively.

Table XIX: Correct Answers Per Problem and Per Participant with Average

Evaluated technique		Category	Participants									Correct Answers per Problem
			A	B	C	D	E	F	G	H	I	
1	Programming Loop	Data processing	●	×	●	●	●	×	●	×	×	5
2	Previous Position Storing	Data processing	●	●	●	●	●	×	●	●	×	7
3	Picture pixel array management	Data display	×	×	●	×	×	×	●	×	×	2
4	Recursion	Data processing	×	●	●	×	●	×	●	×	×	4
5	Empty Area Detection	Data display	×	●	●	×	×	×	●	●	●	5
6	Hidden Line Removal	Data display	×	●	●	●	●	×	×	●	×	5
7	Distribute Objects According To Data	Data display	×	●	●	●	×	●	●	●	×	6
8	Multiple Data Sources	Data processing	×	×	●	●	×	×	×	●	×	3
9	Visualization According to Data	Data display	●	●	×	×	●	●	×	●	×	5
10	Data Format Reading	Data processing	×	●	●	×	●	●	●	●	×	6
Correct Answers per Participant			3	7	9	5	6	3	7	7	1	AVG : 5

On the other hand, 2 of the 3 participants who reported having the less professional and programming experience obtained the lowest score: Participants A and F each one with less than 1 year of general experience and 1-3 years of programming experience answered correctly to 3 problems.

These results confirm our assumption since the more experienced developers only failed two or three problems including those that received the least correct answers (picture pixel array management with 2 correct answers and multiple data sources with 3).

In contrast the less experienced programmers only answered correctly one to three problems, however, the participant F was able to answer two difficult data display problems and one difficult data processing problem; this rather contradictory result may be due to his background, this participant reported being a content developer, and having managed JavaScript.

The results of one particular participant stand out for being an exception to our hypothesis: participant B who, with less than one year of general experience, no experience in programming and having not reported any knowledge on programming

languages, answered correctly to seven problems. We will explore in more detail this case in section 7.5.

By assigning scores to the answers of the questionnaire on experience and knowledge composed by two questions on knowledge of the evaluated programming technique (yes/no questions) and three questions on experience with the mentioned technique (Likert scale) and use them together with the correct answers per participant per problem we were able to calculate the Pearson correlation coefficient for each of the variables per problem.

We performed also a regression analysis by using the same data to find the Coefficient of Determination (R square) in order to establish if the test score can be predicted by the answers of the questionnaire so we could validate the correlation. For the purposes of this research, problems with an R square coefficient equal or greater than 0.7 were considered significant.

In the following sections we will discuss about the possible reasons why IT experts having a high score on the test failed to answer correctly some of the problems proposed in the PVCC based test, and in addition, what kind of programming knowledge and experience do those IT experts answering to the test possess, and how does these aspects relate with their answers to the sample comparisons in general terms and per significant problems.

7.4. Why do IT Experts Fail at Answering Problems Based on the PVCC

Method?

From Table XIX, it can be seen that, within the set of 10 problems, 3 of them were answered correctly by less than 5 people, namely those problems evaluating the techniques: *Picture pixel array management*, *Multiple data sources* and *Recursion*, which were answered correctly by 2, 3 and 4 people respectively.

It is important to mention that, all three problems were answered correctly only by the participant with the highest score, or participant C with 9 total correct answers, but of the three people that followed him in the score table, namely, participants B, G and H each one with 7 total correct answers, two of them failed two of the aforementioned problems.

As it can be seen in Table XVII, excluding participant B who is the less experienced of all, participants G and H are both experienced in infrastructure and

development but one of them with more than 10 years of experience, and both have knowledge in programming, so it is valid to ask why did they answered incorrectly these particular problems?

A possible explanation for these results might be that, even when guidance resources like a hint, instructions on how to operate and what kind of content was shown in each sample comparison were added, there were issues at interpreting, reading or understanding these problems, therefore being still misleading.

Let us now analyze more closely the worst performing of these three problems to try to understand why is it still misleading. We are going to focus specially on how the hint and the operation instructions guide the participant towards a supposedly correct interpretation of the problem because there could be the main weakness of these problems.

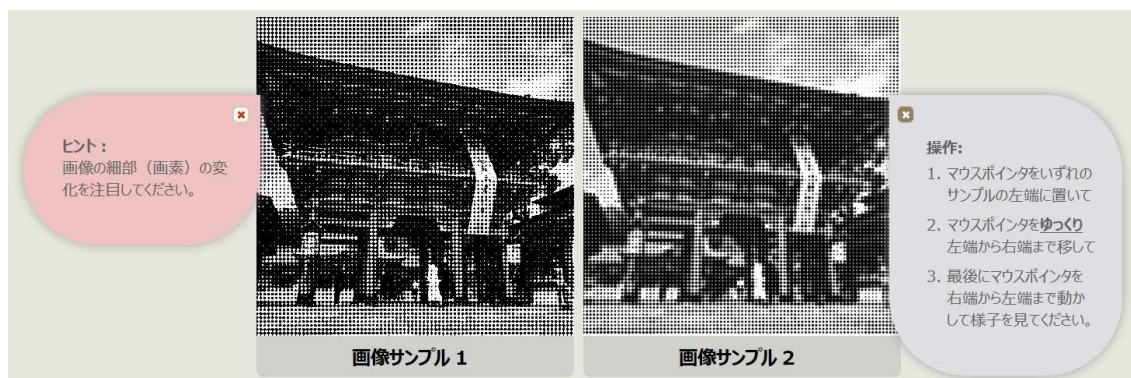


Fig. 21 Appearance of the problem evaluating the technique: *Picture pixel array management*

Fig. 21 shows the appearance of problem evaluating the technique: *Picture pixel array management*; for the first sample of this problem, when the mouse is moved horizontally the color of each pixel is rewritten according to a filter that changes in a uniform way the color of each pixel without considering dark or light values; on the other hand, for the second sample the values of darkness and lightness of the picture are identified and mapped to circles size, so certain small circles represent light areas and bigger circles represent dark areas; if the mouse is moved horizontally, the dark and light values changes proportionally, so does the size of each circle.

The hint for this sample comparison was: “pay attention to how the pixels change on each picture!”, and the operation instructions were as follows:

- 1) Place the mouse over the left border of any of the samples
- 2) Move the mouse SLOWLY from the left border to the right border of the sample.

- 3) Finally, move the mouse pointer from the right border to the left border and see what happens!

Table XX: Answers to Sample Comparison and Knowledge and Experience Questions for the Technique: *Picture Pixel Array Management*

Technique: Picture pixel array management	Participants								
	A	B	C	D	E	F	G	H	I
Answers to problem	sample 1	sample 1	sample 2	sample 1	similar	skipped	sample 2	similar	sample 1
Programming knowledge questions									
After reading the previous explanation did you understand how “picture pixel array management” is used in this problem?	no	yes	yes	no	no	skipped	yes	no	no
Did you realize about the use of “picture pixel array management” on this problem before the explanation?	no	yes	yes	no	no	skipped	yes	yes	no
Programming experience questions									
Have you ever applied “filters” (or options to manipulate the pixels of an image) to pictures by using software tools? (Microsoft Paint, Adobe Photoshop, etc.)	never	I'm used to do it	once or twice	many times	never	skipped	never	never	once or twice
Have you ever called or referenced external pictures to show them on screen with programming?	never	never	once or twice	never	never	skipped	once or twice	once or twice	once or twice
Have you ever programmed filters for pictures directly with programming? (to change picture properties like saturation, contrast, brightness etc.)	never	never	once or twice	never	never	skipped	never	never	never

Table XXI: Correlation Coefficients for Answers to the Sample Comparison and Knowledge and Experience Questions for the Problem Evaluating: *Picture Pixel Array Management*

Picture pixel array management	Problem	Knowledge 1	Knowledge 2	Experience 1	Experience 2	Experience 3
Problem	1.00					
Knowledge 1	0.76	1.00				
Knowledge 2	0.60	0.79	1.00			
Experience 1	-0.14	0.38	0.19	1.00		
Experience 2	0.60	0.32	0.55	-0.24	1.00	
Experience 3	0.66	0.50	0.40	0.08	0.40	1.00

Table XX shows the answers to the problem and the questionnaire about experience and knowledge in programming for the evaluated technique provided by each one of the participants, and table XXI shows the correlation coefficients for these answers,

The R square coefficient for the results of this problem confirmed through a regression analysis was **0.91**. The correlation coefficients shown on table XXI indicate that the understanding of the explanation and the awareness of the use of the technique before reading it, together with the answers to the two questions about experience with

the technique evaluated in programming (not software tools) are closely correlated to the provided problem answer, therefore those experienced with the technique in programming, who understood the explanation and were aware of the technique without reading the explanation most likely answered the sample comparison correctly and vice versa.

Closer inspection to table XXI shows that participant B, even when understanding the explanation and being aware of the technique, answered by selecting the opposite sample; in addition, participant H, who is the most experienced person in programming among this group, answered incorrectly to the sample comparison and reported not having understood the explanation provided after answering the problem.

The aforementioned facts suggest that being aware of the use of the technique in the problem is not enough to answer correctly and the participant needs to understand almost in detail how the technique is managed in this particular problem (probably by having dealt with it before), or in the other hand, considering that, according to the correlation analysis, those who have experience and were aware of the use of the technique even before reading the explanation almost certainly answered correctly, it may be that those participants who surely knew the answer (i.e. participants B and H) were in fact misguided by the additional guidance elements.

In this sense, it is necessary to correct strictly the guidance elements, namely the hint and the operation guide, so they don't suggest a different interpretation than the one required for the problem, and verify that the relation between these elements and the correct answer is coherent for the participant to be really guided to the answer.

7.5. What Kind of Programming Knowledge and Experience Do IT Experts Have?

This thesis finishes with a thorough discussion about what is the programming knowledge and experience of those participants answering right and wrong to the problems about programming techniques, and how their knowledge of and experience with each programming technique correlates with the provided answers, highlighting those problems that we consider representative, and also their representative answers, focusing particularly on what kind of individuals (with what experience and knowledge) provided those answers.

Table XXII provides an overview on the total amount of correct and incorrect answers to the sample comparison problems and the respective total answers to the questions on knowledge of each technique and on experience with each technique.

Table XXII: Overview on The Total of Correct and Incorrect Answers to Problems and Answers to Questions on Knowledge and Experience with Evaluated Techniques

Total of Correct Answers to Sample Comparisons			48	
Questions on Knowledge of each Programming Technique	Understands explanation	yes	42	
		no	6	
	Was aware of technique	yes	39	
		no	9	
Questions on Experience with each Programming Technique	Experience with Software tools	I'm used to do it	13	
		many times	13	
		once or twice	11	
	Experience with Programming	never	11	
		I'm used to do it	20	
		many times	2	
	Experience with Programming	once or twice	13	
		never	13	
		I'm used to do it	12	
		many times	5	
		once or twice	7	
		never	24	

Total of Incorrect Answers to Sample Comparisons			42	
Questions on Knowledge of each Programming Technique	Understands explanation	yes	17	
		no	19	
	Was aware of technique	skipped	6	
		yes	15	
Questions on Experience with each Programming Technique	Experience with Software tools	no	21	
		skipped	6	
		I'm used to do it	5	
	Experience with Programming	many times	3	
		once or twice	10	
		never	18	
	Experience with Programming	skipped	6	
		I'm used to do it	5	
		many times	1	
		once or twice	12	
		never	18	
		skipped	6	
Experience with Programming	Experience with Programming	I'm used to do it	4	
		many times	0	
		once or twice	6	
	never	26		
	skipped	6		

As shown in table XXII the number of correct answers to the problems was bigger than the number of incorrect answers only by a little margin, but by looking to the number of answers to questions on knowledge of the technique for the problems having *incorrect* answers to sample comparisons, it can be seen that the number of participants that understood the explanation and was aware of the technique evaluated before answering was scarcely smaller than those not understanding it and not being aware of the technique.

If these results are compared with the number of answers to the questions about experience, that show that the majority of participants have applied the technique once or twice or have never used the technique. Oneither in software tools or with programming, the whole of these results may suggest that many of the participants who reportedly have knowledge of these techniques may in fact have very limited or not any experience with them at all.

It may be the case therefore, that not only knowledge, but at least some experience with the technique is needed to answer correctly; this could be perceived to some extent

by examining the number of answers to questions on knowledge and experience for the problems having *correct* answers to sample comparisons also shown in Table XXII.

A significant majority of participants answering correctly to problems with sample comparisons reported understanding the explanation provided for the techniques evaluated and besides being aware of the technique before answering; In addition, more than half of the participants reported having applied these techniques many times or being used to do them in software tools, and almost half of them also reported having experienced these techniques many times or mastered them in programming.

Until now this section has provided an overview and a primary analysis on the results of the test carried on with IT experts, that allowed us to understand that, at least for this group of participants, not only knowledge on a programming technique but solid experience on it is necessary to answer correctly a question based on the PVCC method.

It is now necessary to verify the correlation between experience and knowledge and the answers to sample comparisons by technique evaluated and highlight what we consider are representative cases that contribute directly to accomplish the objective described in section 7.1. we will exclude those problems that could have presented issues, explained in section 7.4. as well as those obtaining an insignificant R square correlation coefficient.

7.5.1. Programming Loop

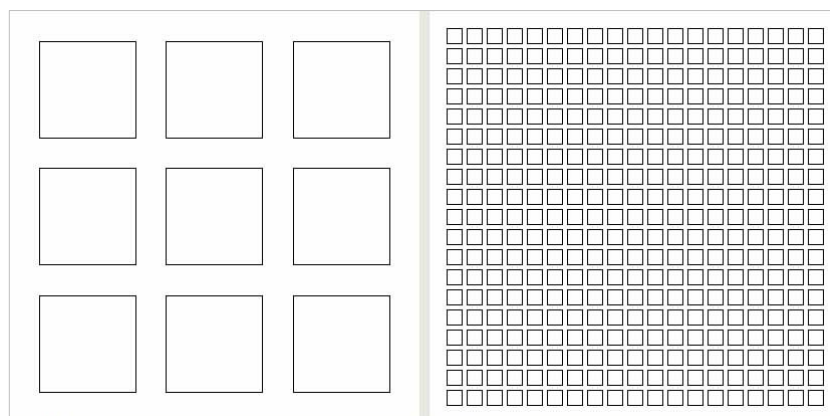


Fig. 22 Sample comparison evaluating the technique: *Programming Loop*

The R square coefficient for the results of this problem confirmed through a regression analysis was **0.88** and, as shown in Table XIX this problem was answered correctly by 5 participants therefore being within the average of correct answers per problem. Table XXIII shows the answers to the problem and the questionnaire about

experience and knowledge in programming for the evaluated technique provided by each one of the participants, and table XXIV shows the correlation coefficients for these answers.

Table XXIII: Answers to Sample Comparison and Knowledge and Experience Questions for the Technique: *Programming Loop*

Technique: Programming Loop	Participants								
	A	B	C	D	E	F	G	H	I
Answers to problem	similar	skipped	similar	similar	similar	skipped	similar	sample 2	sample 1
Programming knowledge questions									
After reading the previous explanation did you understand how “programming loop” is used in this problem?	yes	skipped	yes	yes	yes	skipped	yes	yes	no
Did you realize about the use of “Programming Loop” on this problem before reading the explanation?	no	skipped	yes	yes	yes	skipped	yes	yes	yes
Programming experience questions									
Have you ever used automatic actions on software tools (ex: the “redo” button on word, or “recorded actions” on photoshop)?	once or twice	skipped	many times	I'm used to do it	I'm used to do it	skipped	many times	once or twice	once or twice
Have you ever used loops in programming? (keywords: for, while)	once or twice	skipped	I'm used to do it	I'm used to do it	I'm used to do it	skipped	I'm used to do it	I'm used to do it	once or twice
Have you ever written code where you had to use loops inside loops (or nested loops) in programming?	once or twice	skipped	many times	I'm used to do it	many times	skipped	I'm used to do it	I'm used to do it	never

Table XXIV: Correlation Coefficients for Answers to the Sample Comparison and Knowledge and Experience Questions for the Problem Evaluating: *Programming Loop*

Programming loop	Problem	Knowledge 1	Knowledge 2	Experience 1	Experience 2	Experience 3
Problem	1.00					
Knowledge 1	0.79	1.00				
Knowledge 2	0.32	0.50	1.00			
Experience 1	0.79	0.74	0.74	1.00		
Experience 2	0.62	0.86	0.86	0.85	1.00	
Experience 3	0.57	0.88	0.69	0.73	0.93	1.00

The correlation coefficients shown on table XXIV indicate that the understanding of the explanation, and all the answers to the questions about programming experience with the technique evaluated, but particularly the question about experience with the technique on software tools, are closely correlated to the provided problem answer, so those experienced in at least the three aspects of the questionnaire, particularly in software tools, who understood the explanation most likely answered the sample comparison correctly and vice versa. The correlation analysis confirms that participants’ experience and knowledge of the evaluated technique: Programming Loop are strongly

correlated with the answer to the sample comparison for the participant group of IT experts.

Closer inspection to table XXIII shows that participant H, instead of identifying the similar difficulty of both sample opted for selecting one of them, even when he understood the explanation, he reported enough experience on programming with this technique, he is one of the most experienced developers and considering that the hint for this problem clearly indicated that the number of squares wasn't important. This may suggest that there is an aspect of the technique evaluated with this sample comparison, beyond the number of squares, perceived by experienced programmers that could be making them choose the second sample; this is an unexpected result that could be the base for a revision of this particular problem.

Table XXIII also indicates that two participants skipped the question, they may have been confused enough to not find an answer to this sample comparison; this is a possibility we have to consider for the rest of the problems.

7.5.2. Recursion

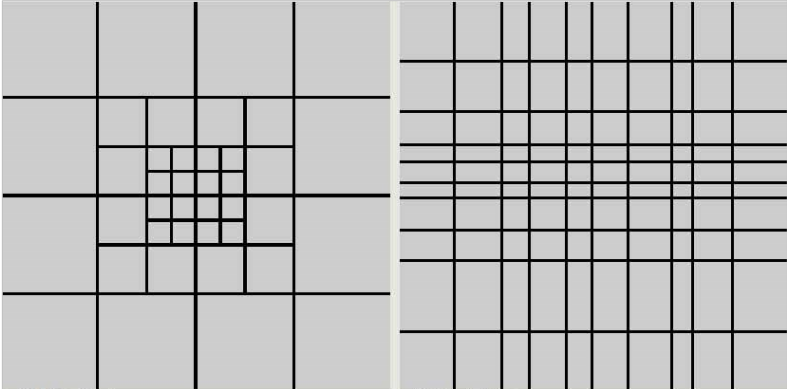


Fig. 23 Sample comparison evaluating the technique: *Recursion*

The R square coefficient for the results of this problem confirmed through a regression analysis was **0.77** and, as shown in Table XIX this problem was answered correctly by 4 participants out of 9 therefore being below the average of correct answers per problem. Table XXV shows the answers to the problem and the questionnaire about experience and knowledge in programming for the evaluated technique provided by each one of the participants, and table XXVI shows the correlation coefficients for these answers.

Table XXV: Answers to Sample Comparison and Knowledge and Experience Questions for the Technique: *Recursion*

Technique: Recursion	Participants								
	A	B	C	D	E	F	G	H	I
Answers to problem	sample 2	sample 1	sample 1	sample 2	sample 1	skipped	sample 1	skipped	sample 2
Programming knowledge questions									
After reading the previous explanation did you understand how “Recursion” is used in this problem?	no	yes	yes	yes	yes	skipped	yes	skipped	yes
Did you realize about the use of “Recursion” on this problem before the explanation?	no	yes	yes	yes	no	skipped	yes	skipped	no
Programming experience questions									
Have you ever used elements in software tools that can contain themselves (for example: in word and powerpoint: tables can contain tables, or textboxes can contain textboxes)?	never	I'm used to do it	never	never	never	skipped	never	skipped	never
Have you ever done mathematic exercises or practices about mathematical recursive sequences (for example: the Fibonacci sequence)?	never	never	many times	never	never	skipped	I'm used to do it	skipped	never
Have you ever used or created recursive functions in programming?	never	never	once or twice	never	never	skipped	once or twice	skipped	never

Table XXVI: Correlation Coefficients for Answers to the Sample Comparison and Knowledge and Experience Questions for the Problem Evaluating: *Recursion*

Recursion	Problem	Knowledge 1	Knowledge 2	Experience 1	Experience 2	Experience 3
Problem	1.00					
Knowledge 1	0.63	1.00				
Knowledge 2	0.55	0.63	1.00			
Experience 1	0.40	0.25	0.40	1.00		
Experience 2	0.58	0.37	0.58	-0.18	1.00	
Experience 3	0.60	0.38	0.60	-0.19	0.98	1.00

The correlation coefficients shown on table XXVI indicate that the understanding of the explanation, together with the two questions about programming experience with the technique evaluated are correlated to the provided problem answer, therefore those experienced in the technique at least in the two aspects related with programming, who understood the explanation most likely answered the sample comparison correctly and vice versa.

The correlation analysis confirms that the experience and knowledge on the evaluated technique: Recursion are strongly correlated with the answer to the sample comparison, regarding the participant group of IT experts.

The most interesting data shown by table XXV is the result of participant E; this participant has never used recursion in programming, and was not familiarized with the subject from mathematics; he hasn't even used the options more related with the technique when referring to software tools but his knowledge of the recursion technique matched with the one provided in the explanation so when he read it he understood how the technique was applied.

This participant is the most experienced individual in systems operation and management of the group but has only 1 to 3 years of programming experience; according to the aforementioned facts we can infer that his knowledge about Recursion could be field-specific knowledge, not related with programming nor with software tools.

The aforementioned case could be a good example of a subject who acquired a *Panoramic Understanding* of a *Programming* technique by means of possessing and/or applying knowledge from other fields.

7.5.3. Empty Area Detection

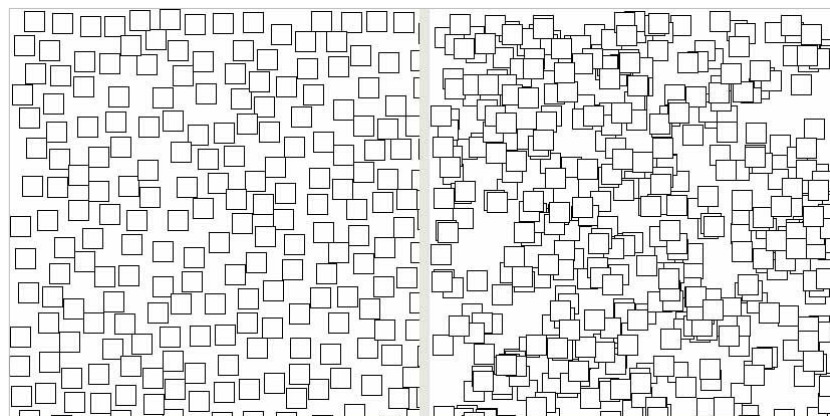


Fig. 24 Sample comparison evaluating the technique: *Empty Area Detection*

The R square coefficient for the results of this problem confirmed through a regression analysis was **0.80** and, as shown in Table XIX this problem was answered correctly by 5 participants out of 9, therefore being within the average of correct answers per problem.

Table XXVII shows the answers to the problem and the questionnaire about experience and knowledge in programming for the evaluated technique provided by each one of the participants, and table XXVIII shows the correlation coefficients for these answers.

Table XXVII: Answers to Sample Comparison and Knowledge and Experience Questions for the Technique: *Empty Area Detection*

Technique: Empty Area Detection	Participants								
	A	B	C	D	E	F	G	H	I
Answers to problem	similar	sample 1	sample 1	sample 2	similar	skipped	sample 1	sample 1	sample 1
Programming knowledge questions									
After reading the previous explanation did you understand how “empty area detection” is used in this problem?	no	yes	yes	yes	yes	skipped	yes	no	no
Did you realize about the use of “empty area detection” on this problem before the explanation?	no	yes	yes	no	no	skipped	yes	no	no
Programming experience questions									
Have you ever used “layers” in software tools? (layers are areas that can contain many objects and can be changed independently of other similar “layers”, you can find them in software like word, powerpoint, photoshop and others)	never	I'm used to do it	once or twice	many times	never	skipped	once or twice	once or twice	never
Have you ever used conditionals in programming (keywords: if, while)	once or twice	never	I'm used to do it	never	many times	skipped	I'm used to do it	I'm used to do it	once or twice
Have you ever drawn elements on screen with programming depending on the result of other processes (for example, drawing many rectangles with a loop)?	once or twice	never	once or twice	never	never	skipped	many times	never	never

Table XXVIII: Correlation Coefficients for Answers to the Sample Comparison and Knowledge and Experience Questions for the Problem Evaluating: *Empty Area Detection*

Empty Area Detection	Problem	Knowledge 1	Knowledge 2	Experience 1	Experience 2	Experience 3
Problem	1.00					
Knowledge 1	0.10	1.00				
Knowledge 2	0.63	0.63	1.00			
Experience 1	0.35	0.58	0.55	1.00		
Experience 2	0.49	0.14	0.31	-0.23	1.00	
Experience 3	0.25	0.25	0.57	-0.09	0.54	1.00

The correlation coefficients shown on table XXVIII indicate that the awareness of the use of the technique before reading the explanation, together with the question about programming experience with conditionals, that is the base for the technique evaluated in this problem, are correlated to the provided problem answer, therefore, those experienced in the technique at least in the aspect asked who were aware of the technique since the beginning surely answered the problem correctly and vice versa.

The correlation analysis confirms that the experience and knowledge on the evaluated technique: Empty Area Detection are in some aspects correlated with the answer to the sample comparison.

The most interesting aspect of table XXVII is that participant B, mentioned before as an exceptional case, without any experience on the concept base of the technique evaluated: *conditionals* necessary to perform the empty area detection, nor on making objects appear on screen depending on other programming processes, and with only experience on the concept of *layers* in software tools was aware of the empty area detection before answering the sample comparison and was able to understand our explanation. This participant is another example of an individual who acquired a *panoramic understanding* of complex *programming* techniques by applying knowledge from other fields and by using only software tools.

7.5.4. Distribute Objects According to Data



Fig. 25 Sample comparison evaluating the technique: *Distribute Objects According to Data*

The R square coefficient for the results of this problem confirmed through a regression analysis was **1.0** and, as shown in Table XIX this problem was answered correctly by 6 participants out of 9, therefore being above the average of correct answers per problem. Table XXIX shows the answers to the problem and the questionnaire about experience and knowledge in programming for the evaluated technique provided by each one of the participants, and table XXX shows the correlation coefficients for these answers.

The correlation coefficients shown on Table XXX indicate that, particularly, the understanding of the explanation provided but also the awareness of the use of the

technique before reading the explanation, together with the question about changing shapes properties with code are strongly correlated to the problem answer, therefore, those participants experienced at least in the aspect related with programming already mentioned, who were aware of the technique before reading the explanation but specially whose knowledge on the technique was almost the same to that provided in the explanation, most likely answered the problem correctly and vice versa.

The correlation analysis confirms that, remarkably, the knowledge on the evaluated technique: Distribute Objects According to Data but also some aspects of the experience are strongly correlated with the answer to the sample comparison.

What stands out in table XXIX is that participant F, who skipped four of the problems previous to this one and answered incorrectly other two, demonstrated in this problem having knowledge and experience in this particular technique and answered the problem correctly.

Table XXIX: Answers to Sample Comparison and Knowledge and Experience Questions for the Technique: *Distribute Objects According to Data*

Technique: Distribute Objects According to Data	Participants								
	A	B	C	D	E	F	G	H	I
Answers to problem	sample 2	sample 1	sample 1	sample 1	similar	sample 1	sample 1	sample 1	sample 2
Programming knowledge questions									
After reading the previous explanation did you understand how “distribute objects according to data” is used in this problem?	no	yes	yes	yes	no	yes	yes	yes	no
Did you realize about the use of “distribute objects according to data” on this problem before the explanation?	no	yes	yes	yes	no	yes	yes	yes	yes
Programming experience questions									
Have you ever used the “align and distribute” option that makes objects align to center, left or right and distribute the space between them in software tools? (Example: Microsoft powerpoint, photoshop etc.)	I'm used to do it	I'm used to do it	once or twice	never	once or twice	I'm used to do it	I'm used to do it	many times	never
Have you ever changed the properties of shapes (size, color, position etc) depending on data located in external files with programming?	never	never	once or twice	I'm used to do it	never	I'm used to do it	once or twice	once or twice	never
Have you ever wrote code to align and distribute shapes or objects in a pattern (ex: distribute shapes in a circle pattern or in a line)	never	never	once or twice	never	never	I'm used to do it	once or twice	never	never

Table XXX: Correlation Coefficients for Answers to the Sample Comparison and Knowledge and Experience Questions for the Problem Evaluating: *Distribute Objects According to Data*

Distribute Objects According to Data	Problem	Knowledge 1	Knowledge 2	Experience 1	Experience 2	Experience 3
Problem	1.00					
Knowledge 1	1.00	1.00				
Knowledge 2	0.76	0.76	1.00			
Experience 1	0.26	0.26	-0.10	1.00		
Experience 2	0.61	0.61	0.46	-0.08	1.00	
Experience 3	0.41	0.41	0.31	0.39	0.60	1.00

This is an unexpected outcome for the general results of this problem, and most likely suggests that this was one of the only problems where he, as a content developer, found his background and experience useful to answer correctly.

Another aspect to highlight in table XXIX is, once more, the results and answers of participant B who having only experienced shape distribution in software tools, was aware of the technique before answering the sample comparison and his knowledge on the technique was similar to the explanation we provided.

7.5.5. Visualization According to Data

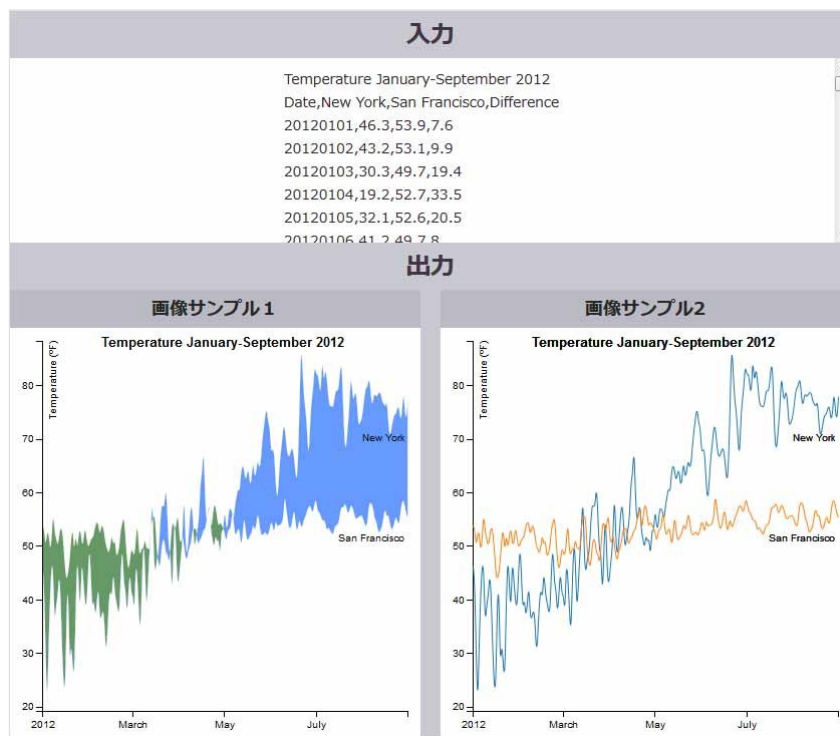


Fig. 26 Sample comparison evaluating the technique: *Visualization According to Data*

The R square coefficient for the results of this problem confirmed through a regression analysis was 0.75 and, as shown in Table XIX this problem was answered

correctly by 5 participants out of 9, therefore being within the average of correct answers per problem.

Table XXXI shows the answers to the problem and the questionnaire about experience and knowledge in programming for the evaluated technique provided by each one of the participants, and table XXXII shows the correlation coefficients for these answers.

The correlation coefficients shown on Table XXXII indicate that, only the question about experience with the technique in software tools is strongly correlated to the problem answer, therefore, those participants who have made polygons with lines by using software tools most likely answered correctly to the sample comparison and vice versa.

The correlation analysis confirms that at least one experience aspect is strongly correlated with the answer to the sample comparison for the technique evaluated.

Table XXXI: Answers to Sample Comparison and Knowledge and Experience Questions for the Technique: *Visualization According to Data*

Technique: Visualization According to Data	Participants								
	A	B	C	D	E	F	G	H	I
Answers to problem	sample 1	sample 1	similar	sample 2	sample 1	sample 1	similar	sample 1	similar
Programming knowledge questions									
After reading the previous explanation did you understand how “data visualization according to data analysis” is used in this problem?	yes	yes	yes	yes	yes	yes	yes	no	no
Did you realize about the use of “data visualization according to data analysis” on this problem before the explanation?	yes	yes	yes	no	no	yes	no	no	yes
Programming experience questions									
Have you ever made polygons with lines by using software tools (for example in powerpoint you can join together lines to build a polygon then you can change its shape from its vertices or apply color inside)	once or twice	many times	once or twice	never	many times	I'm used to do it	once or twice	many times	never
Have you ever calculated the area between two curves in mathematics?	never	never	never	never	once or twice	I'm used to do it	once or twice	never	never
Have you ever calculated the area between two curves with programming?	never	never	never	never	never	I'm used to do it	once or twice	never	once or twice

Table XXXII: Correlation Coefficients for Answers to the Sample Comparison and Knowledge and Experience Questions for the Problem Evaluating: *Visualization According to Data*

Visualization According to Data	Problem	Knowledge 1	Knowledge 2	Experience 1	Experience 2	Experience 3
Problem	1.00					
Knowledge 1	0.06	1.00				
Knowledge 2	0.10	0.06	1.00			
Experience 1	0.79	0.19	0.08	1.00		
Experience 2	0.29	0.31	0.05	0.66	1.00	
Experience 3	0.05	0.03	0.29	0.41	0.88	1.00

It is apparent from the correlation coefficients shown on table XXXII and the data on table XXXI that the questions about experience and knowledge for this problem need to be changed, this problem is not related with calculation of the area below the curve nor in mathematics neither in programming. The explanation also needs to be improved because some participants (specifically participants C, D and G) even when understanding this text and one of them having reported that was aware of the technique evaluated from the beginning, answered incorrectly to the sample comparison, these findings suggest that probably the explanation is too general and doesn't give much real detail on how the technique evaluated is applied in the correct sample.

7.5.6. Data Format Reading

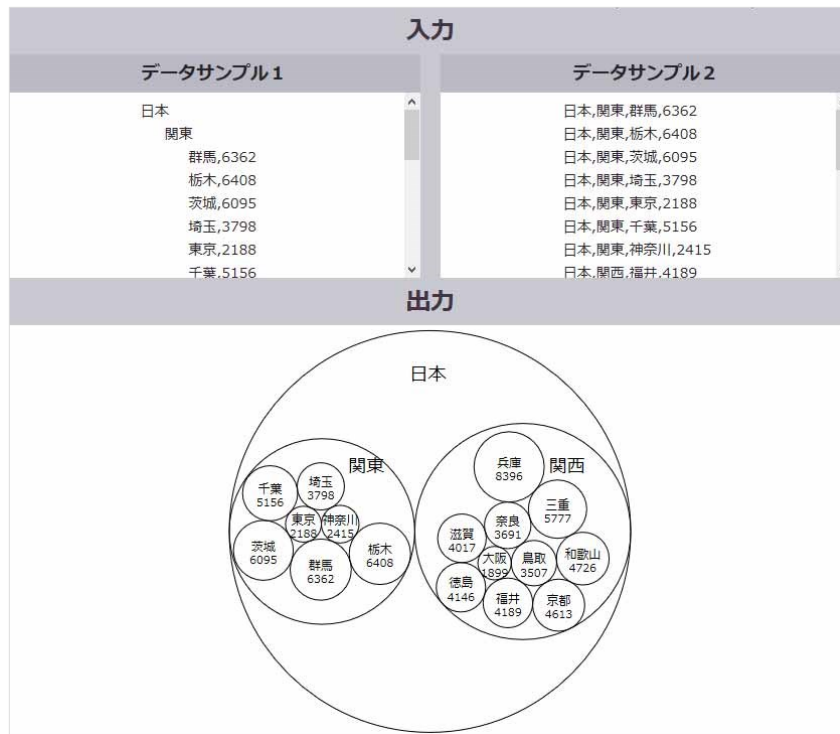


Fig. 27 Sample comparison evaluating the technique: Visualization According to Data

The R square coefficient for the results of this problem confirmed through a regression analysis was 1.0 and, as shown in Table XIX this problem was answered correctly by 6 participants out of 9, therefore being above the average of correct answers per problem.

Table XXXIII shows the answers to the problem and the questionnaire about experience and knowledge in programming for the evaluated technique provided by each one of the participants, and table XXXIV shows the correlation coefficients for these answers.

The correlation coefficients shown on Table XXXIV indicate that, notably, the understanding of the explanation provided but also the awareness of the use of the technique before reading the explanation, together with the question on programming experience regarding different formats' data reading with code are strongly correlated to the problem answer, therefore, those participants experienced at least in the aspect related with programming already mentioned, who were aware of the technique before reading the explanation but particularly whose knowledge on the technique was almost the same to that provided in the explanation, surely answered the problem correctly and vice versa.

Table XXXIII: Answers to Sample Comparison and Knowledge and Experience Questions for the Technique: *Data Format Reading*

Technique: Data Format Reading	Participants								
	A	B	C	D	E	F	G	H	I
Answers to problem	sample 2	sample 1	sample 1	sample 2	sample 1	sample 1	sample 1	sample 1	sample 2
Programming knowledge questions									
After reading the previous explanation did you understand how “data text format reading and displaying” is used in this problem?	no	yes	yes	no	yes	yes	yes	yes	no
Did you realize about the use of “data text format reading and displaying” on this problem before the explanation?	no	yes	yes	no	yes	yes	yes	yes	yes
Programming experience questions									
Have you ever transformed spreadsheets into csv (comma separated value) files? (for example: in excel you can transform a table to a comma separated value file)?	I'm used to do it	never	I'm used to do it	never	many times	I'm used to do it	I'm used to do it	many times	never
Have you ever opened text files or a comma separated value (csv) files in text processors or spreadsheet software (ex: word or excel)?	once or twice	once or twice	I'm used to do it	never	many times	I'm used to do it	I'm used to do it	I'm used to do it	I'm used to do it
Have you ever done a program to read data from different text formats?	never	never	I'm used to do it	never	many times	I'm used to do it	I'm used to do it	many times	once or twice

Table XXXIV: Correlation Coefficients for Answers to the Sample Comparison and Knowledge and Experience Questions for the Problem Evaluating: *Data Format Reading*

Data Format Reading	Problem	Knowledge 1	Knowledge 2	Experience 1	Experience 2	Experience 3
Problem	1.00					
Knowledge 1	1.00	1.00				
Knowledge 2	0.76	0.76	1.00			
Experience 1	0.42	0.42	0.11	1.00		
Experience 2	0.50	0.50	0.78	0.48	1.00	
Experience 3	0.69	0.69	0.66	0.68	0.84	1.00

The correlation analysis confirms that knowledge related to the evaluated technique: Data Format Reading and also some aspects of the experience are strongly correlated with the answer to the problem.

Answers to this problem provided by participant B stand out once again as representative of an individual who possess a Panoramic Understanding of the programming processes needed to read data formats and the difficulty that it implies; this participant was aware of the difference on text formats of the data samples provided and how this difference can affect program's behavior; additionally he was able to identify which one of the formats would be more difficult to read for a program therefore answering correctly to the sample comparison.

7.6. Conclusion of This Chapter

Through the consolidation of corrected problems from the two previous tests and the addition of a questionnaire on experience and knowledge in programing and software tools to each problem, we were able to build a new test based on the PVCC method and apply it to experts in information and technology obtaining results which analysis provided detiled insights on the relation between knowledge and experience in programming and the panoramic understanding of a programming technique.

Taken together the results and the analysis of the test presented in this section suggest that, at least for the group of IT experts evaluated, there is an association between the experience and knowledge (understanding and awareness of what is being evaluated) for the programming techniques evaluated and the answers to the programming sample comparison problems.

In other words: if an individual of the participant group, knowing the evaluated technique in beforehand by having managed it in software tools or by applying it while doing programming, answers to one of the questions of the PVCC method based test, he

most likely will be aware of the technique before answering and then answer correctly, and vice versa, if the participant don't know the technique enough to be able to be aware of it when looking at the programming samples, he may not be able to answer correctly.

These results confirm our assumption and, in general, contribute to our understanding of how a person could generate a Panoramic Understanding of a programming technique by having knowledge and experience on it not only as an expert but as a beginner or by only having managed software tools.

At the same time from the analysis of the results of this test we were also able to understand that some problems have misleading guidance elements that could have confused IT experts on what is the main purpose of each one of those problems, it is necessary to revise guidance elements, namely the hint and the operation guide, of these problems so they don't suggest a different interpretation than the one desired.

8. Conclusions

This research set out to propose and describe a *Programmed Visual Contents Comparison (PVCC) Method* and determine its potential to evaluate programming abilities related with a *Panoramic Understanding of Programming (PUP)*.

Various experiments involving novice students of programming, IT company staff members, expert programmers from different fields and professors of programming were performed; the results of these experiments confirmed that the programming abilities assessed with the PVCC method are related to a *Panoramic Understanding of Programming*, and the level and difference on knowledge and experience in programming of the person who answers, the difficulty degree of each problem, and how the programming samples are built and paired into a problem define how well this method can evaluate these programming skills.

In the same way, we were able to find other kinds of abilities used by the individuals taking the tests to answer the problems proposed; those abilities, even different to the ones we wanted to assess, are also related with a *Panoramic Understanding of Programming*.

Issues with problems' difficulty identification, criteria for selecting and applying adequate programming subjects at making and pairing programming samples and balance between complexity and concision for each problem were addressed.

Through the addition of input data to comparisons of output sample pictures we were able to build new problems to identify programming abilities. Even when some of the proposed new problems presented different kind of misleading elements, expert programmers and professors participating in the test with these problems identified mistakes and points of confusion and suggested optimal ways to correct them.

By consolidating new sample comparisons and adding questions about experience and knowledge in evaluated programming techniques we could also establish the correlation that the experience and knowledge of the evaluated programming techniques have with the answers provided to the problems based in the PVCC method.

Additionally, through results from feedback questionnaires performed at the end of each test we learned that the test was enjoyable and has a strong potential to evaluate programming abilities; also, that it could become a good starting point for novice

programming learners, or to evaluate programming abilities of new employees at IT companies.

Participants manifested positive feedback regarding the problems and the PVCC method in general. They thought that this method can definitely be used to evaluate programming abilities, and, through the arrangement and correction of problems and the inclusion of questions about experience and knowledge as proposed in this thesis, there is a high probability of evaluating more and more precisely programming abilities related to PUP.

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 individuals' specific programming abilities could produce interesting findings that account more for the validation of the proposed method.

Future improvement should also focus on building evaluation standards or scales for each measured ability related with *Panoramic Understanding of Programming*, and enhance problem's classification, probably proposing different types of tests reaching different level of abilities for different fields or curriculums at programming courses.

A natural progression of this work is to perform more tests using each time more complete and precise problems and keep verifying their effectiveness. Future trials of the test based on the *Programmed Visual Contents Comparison Method* will be carried on with groups of novice and expert programmers from different companies focusing specially in their difference of background and profile and analyzing if these aspects affect or change the way *Panoramic Understanding of Programming* is applied to solve programming problems.

References

- [1] Association for Computing Machinery (ACM); Association for Information Systems (AIS);, “Curriculum Guidelines for Undergraduate Degree Programs in Information Systems (IS 2010),” Association for Computing Machinery and Association for Information Systems, 2010.
- [2] M. Burnett and B. Myers, “Future of End-user Software Engineering: Beyond the Silos,” *Proceedings of the Future of Software Engineering Conference (FOSE 2014)*, ACM, New York, NY, 2014, pp.201-211.
- [3] S. Y. Park, B. Myers and A. Ko, "Designers' Natural Descriptions of Interactive Behaviors," *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '08)*. IEEE Computer Society, Washington, DC, 2008, pp. 185-188.
- [4] A. Ko, B. Myers and H. H. Aung, "Six Learning Barriers in End-User Programming Systems," *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC '04)*. IEEE Computer Society, Washington, DC, 2004, pp. 199-206.
- [5] Mozilla, “MDN Web Docs: HTML,” Mozilla, [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTML>. [Accessed 25 October 2017].
- [6] World Wide Web Consortium, “Cascading Style Sheets Home Page,” w3.org, [Online]. Available: <https://www.w3.org/Style/CSS/Overview.en.html>. [Accessed 20 October 2017].
- [7] F. Kursat Ozenc, K. Miso, J. Zimmerman, S. Oney and B. Myers, "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)*, ACM, New York, NY, 2010, pp. 2513-2522.
- [8] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw and S. Wiedenbeck, “The State of the Art in End-User Software Engineering,” *ACM Computing Surveys*, vol. 43, no. 3, 2011, pp. 1-44.
- [9] D. Norman, *The design of everyday things*, New York, New York: Basic Books, 2013.
- [10] Cycling'74, “About Max,” [Online]. Available: <https://cycling74.com/products/max>. [Accessed 22 october 2017].
- [11] VVVV.org, “VVVV - a multipurpose toolkit,” [Online]. Available: <https://vVVV.org/>. [Accessed 10 october 2017].
- [12] Casa paganini/Infomus - Genova University, “The EyesWeb Project,” [Online]. Available: http://www.infomus.org/eyesweb_eng.php. [Accessed 15 November 2017].
- [13] D. Loksa, A. J. Ko, W. Jerningan, A. Oleson, C. J. Mendez and M. M. Burnett, “Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance,” *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI'16)*, San Jose, California, 2016, pp. 1449-1461.
- [14] T. D. LaToza, D. Garlan, J. D. Herbsleb and B. Myers, “Program Comprehension as Fact Finding,” *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007)*, Dubrovnik, Croatia, 2007, pp. 361-370.

- [15] Qualified Inc., “Codewars.com,” [Online]. Available: <https://www.codewars.com>. [Accessed 12 November 2017].
- [16] Project Euler, “Project Euler.net,” [Online]. Available: <https://projecteuler.net>. [Accessed 15 March 2017].
- [17] Project Nayuki, “Project Nayuki,” [Online]. Available: <https://www.nayuki.io/page/project-euler-solutions>. [Accessed 20 March 2017].
- [18] HackerRank Inc., “HackerRank / Technical Recruiting / Hiring the best Engineers,” [Online]. Available: <https://www.hackerrank.com/>. [Accessed 2 November 2017].
- [19] The University of Aizu, “Aizu Online Judge,” [Online]. Available: <http://judge.u-aizu.ac.jp>. [Accessed 20 November 2017].
- [20] TopCoder Inc., “TopCoder.com: Deliver faster for your business through Crowdsourcing,” [Online]. Available: <https://www.topcoder.com/challenges>. [Accessed 20 November 2017].
- [21] K. Owen, “exercism.io: Level up your programming skills,” [Online]. Available: <http://exercism.io/>. [Accessed 21 November 2017].
- [22] T. LaToza and B. Myers, “On the Importance of Understanding the Strategies that Developers Use,” *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '10)*, ACM, Cape Town, South Africa, 2010, pp. 72-75.
- [23] N. Mangano, T. D. LaToza, M. Petre and A. van der Hoek, “How Software Designers Interact with Sketches at the Whiteboard,” *IEEE Transactions on Software Engineering*, vol. 41, no. 2, 2015, pp. 135 - 156.
- [24] J. Brandt, P. J. Guo, J. Lewenstein and S. R. Klemmer, “Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice,” *Proceedings of the 4th international workshop on End-user software engineering (WEUSE '08)*, Leipzig, Germany, 2008, pp. 1-5.
- [25] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva and S. R. Klemmer, “Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'09)*, Boston, MA, 2009, pp. 1589-1598.
- [26] D. M. Kurland, R. D. Pea, C. Clement and R. Mawby, “A Study of the Development of Programming Ability and Thinking Skills in High School Students,” *Journal of Educational Computing Research*, vol. 2, no. 4, 1986, pp. 429-458.
- [27] R. D. Pea and D. M. Kurland, “On the Cognitive Effects of Learning Computer Programming,” *New Ideas in Psychology*, vol. 2, no. 2, 1984, pp. 137-168.
- [28] B. Myers, J. Pane and A. Ko, “Natural Programming Languages and Environments,” *Communications of the ACM*, vol. 47, no. 9, September 2004, p. 47-52.
- [29] B. Myers, S. Y. Park, Y. Nakano, G. Mueller and A. Ko, “How Designers Design and Program Interactive Behaviors,” *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '08)*, IEEE Computer Society, Washington, DC, 2008, pp. 177-184.
- [30] Adobe Systems Incorporated, “Adobe Muse CC Websites Builder,” Adobe Systems Incorporated, [Online]. Available: <http://muse.adobe.com/>. [Accessed 22 October 2017].

- [31] D. Shiffman, *Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction*, Burlington, MA: Morgan Kaufmann, 2008.
- [32] D. Shiffman, *The Nature Of Code*, New York, NY: Self-published, 2012.
- [33] K. Terzidis, *Algorithms for Visual Design Using the Processing Language*, Indianapolis, IN: Wiley Publishing Inc., 2009.
- [34] H. Bohnacker, B. Gross and J. Laub, *Generative Design: Visualize, Program and Create with Processing*, New York, NY: Princeton Architectural Press, 2012.
- [35] Processing Foundation, "Processing.org," [Online]. Available: <https://processing.org/>. [Accessed 20 October 2017].
- [36] D. Martinez Calderon, Y. Miyamoto, H. Kiyomitsu and K. Ohtsuki, "Characteristics and Advantages of a Visual Contents Comparison Method for Evaluating Programming Abilities (Article in Japanese)" *Proceedings of 2016 Annual Conference of the Institute of Electrical Engineers of Japan C*, 2016, pp. 1317-1319. (D. Martinez Calderon, 宮本 行庸, 清光 英成, 大月 一弘, "視覚コンテンツ比較によるプログラミング能力評価法の特徴と利点," 【C】平成28年電気学会電子・情報・システム部門大会講演論文集, 2016, pp. 1317-1319.)
- [37] D. Martinez Calderon, Y. Miyamoto, H. Kiyomitsu and K. Ohtsuki, "Evaluating Programming Ability by Using a Visual Contents Comparison Method," *Proceedings of the 9th Data Engineering and Information Management Forum (DEIM2017)* (第9回データ工学と情報マネジメントに関するフォーラム(DEIM2017)論文集), 2017, pp. 1-4.
- [38] D. Martinez Calderon, Y. Miyamoto, M. Hirabayashi, H. Kiyomitsu and K. Ohtsuki, "An Evaluation Method for Panoramic Understanding of Programming by Comparison of Programmed Visual Samples," *Information Processing Society of Japan, Computers and Education Research Report* (研究報告コンピューターと教育(CE)), Vols. 2016-CE-134, no. 6, 2016, pp. 1-7.
- [39] D. Martinez Calderon, K. Man, H. Kiyomitsu, K. Ohtsuki, Y. Miyamoto and Y. Sun, "An Evaluation Method for Panoramic Understanding of Programming by Comparison with Visual Examples," *Proceedings of the 2015 IEEE Frontiers in Education Conference (FIE)*, El Paso, Texas, 2015, pp. 1-8.
- [40] D. Martinez Calderon, K. Man, Y. Miyamoto, Y. Sun, M. Hirabayashi, H. Kiyomitsu and K. Ohtsuki, "Measurement Range Increment in a Method for Evaluating Panoramic Understanding of Programming," *Proceedings of the 2016 IEEE Frontiers in Education Conference (FIE)*, Erie, Pennsylvania, 2016, pp. 1-8.
- [41] W. Wang, H. Wang, G. Dai and H. Wang, "Visualization of Large Hierarchical Data by Circle Packing," *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI'06)*, ACM, Montréal, Québec, 2006, pp. 517-520.
- [42] Robert McNeel & Associates, "Rhinceros: design, model, present, analyze, realize," [Online]. Available: <https://www.rhino3d.com/>. [Accessed 24 october 2017].

Acknowledgements

I would like to express my deepest appreciation and thanks to my advisor, Dr. Kazuhiro Ohtsuki, Professor of the Graduate School of Intercultural Studies of Kobe University, for encouraging and allowing me to grow as a researcher. Your advice not only on research but on many other aspects of my career and life has been invaluable.

A very special gratitude goes to Dr. Hidenari Kiyomitsu, Associate Professor of the Graduate School of Intercultural Studies, Computers and Communication Course of Kobe University for all your help during this time; your ideas and support were key to deliver a solid research.

I would especially like to thank all the staff from the College of Computing of Kobe Institute of Computing but particularly to its Vice-president Dr. Kenji Fukuoka, for allowing us to perform this research's initial experiment with the institute students and also for all your ideas and contribution to this research; as well as to Dr. Yukinobu Miyamoto, Associate Professor of the Graduate School of Information Technology of Kobe Institute of Computing for all your continued support and effort during this time, some of the experiments performed in this research in all of its aspects wouldn't have been possible without your help.

I would like to thank as well Dr. Hirokazu Yamamoto, Chairman of the Board and President of ICRAFT Corp. for letting us perform this research's final experiment with ICRAFT's staff members and for allowing them to answer our test during their working time, also for your positive comments and feedback regarding its results.

I am also grateful to Dr. Masami Hirabayashi, Professor of the Institute of Advanced Media Arts and Sciences; Dr. Toshiharu Samura, Professor of the National Institute of Technology, Akashi College; Dr. Ken Bun and Dr. Yi Sun for sharing all their ideas and suggestions during seminars and meetings, and for their technical and personal support during all the process.

This research was possible by a grant from the Japanese Ministry of Education, Culture, Sports, Science and Technology of Japan (Monbukagakusho).

Appendixes

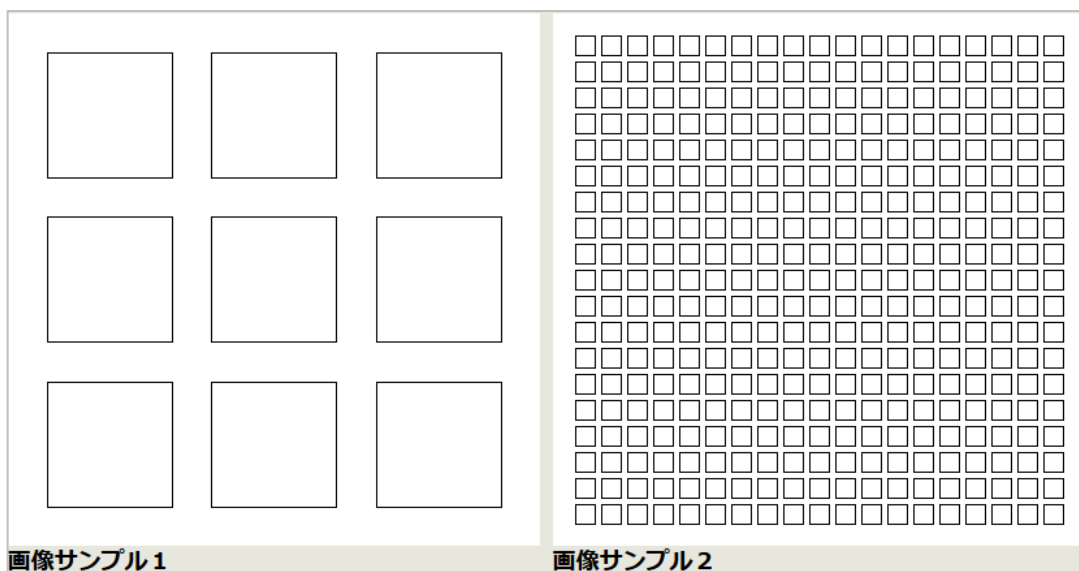
Appendix A: Structure of the Test Based on the PVCC method

This appendix includes sample comparisons explained in section 3, a selection of new problems built as explained in chapter 5 corrected and enhanced according to feedback received as presented in chapter 6 and, in addition, the questionnaire on knowledge and experience in programming presented in section 7.2.2. All problem's questions about knowledge and experience on programming considered were included.

Problem #1

Most Difficult Programming Technique: Programming Loop

Category: Data Processing



Hint: It doesn't matter how many squares there are on each sample or how big they are!

How to handle: There is no need to use the mouse here, just look at the picture in each program sample and answer the question.

Correct Answer: Both samples have similar difficulty.

Why? The first sample seems to be easily achieved by placing lines manually to form squares compared to the second one that can become troublesome because there are many squares to draw, but, both samples have the same code and both squares arrangements

can be quickly performed in programming by drawing the same object (changing its size) with loops.

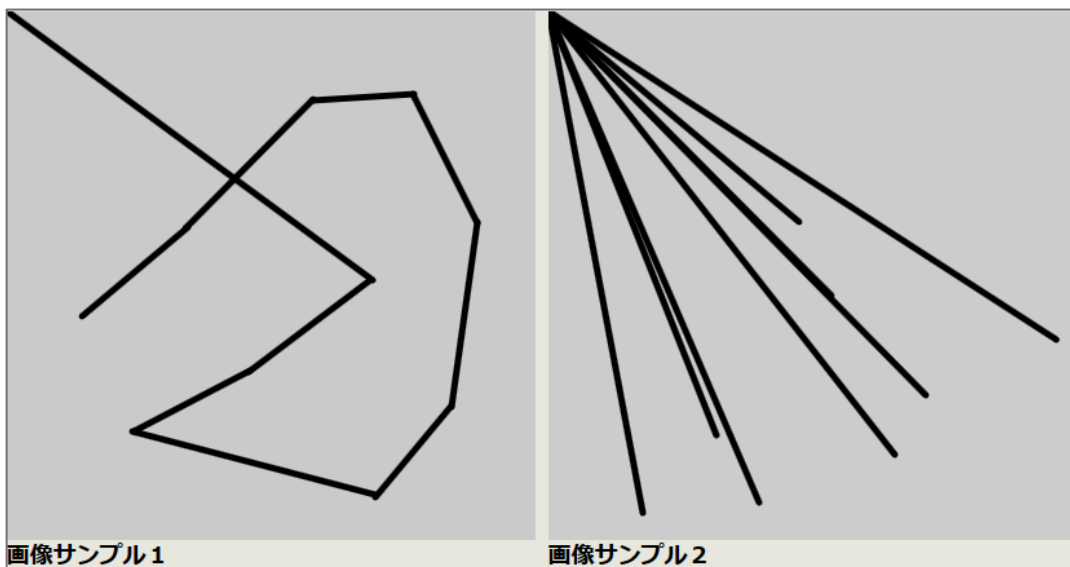
Questions:

- 1) After reading the previous explanation did you understand how “programming loop” is used in this problem?
- 2) Did you realize about the use of “Programming Loop” on this problem before reading the explanation?
- 3) Have you ever used automatic actions on software tools (Ex: the “redo” button on word, or “recorded actions” on photoshop)?
- 4) Have you ever used loops in programming? (keywords: for, while)
- 5) Have you ever written code where you had to use loops inside loops (or nested loops)?

Problem #2

Most Difficult Programming Technique: Previous Position Storing

Category: Data Processing



Hint: pay attention to each line end points!

How to handle:

- 1) Place the mouse over any of the samples
- 2) Press the right Click
- 3) Change the position of the mouse inside the sample, press the right click again and see what happened!

Correct Answer: Sample 1.

Why? The first sample stores the previous mouse position in a variable, and when mouse clicks draws a line between the previous and the current position, while the second sample only draws a line between a fixed point and the mouse position when it's clicked.

Questions:

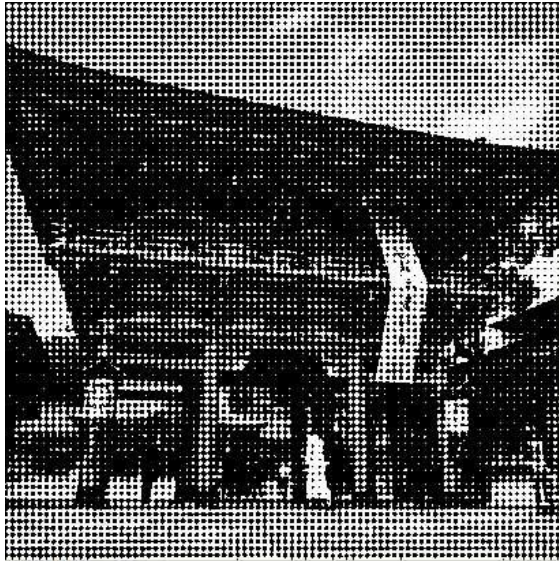
- 1) After reading the previous explanation did you understand how “previous position storing” is used in this problem?
- 2) Did you realize about the use of “previous position storing” on this problem before reading the explanation?
- 3) Have you ever changed the position of shapes by writing numbers or commands instead of using the mouse in software tools? (Ex: change the size of shapes in PowerPoint or Word by using menu options)
- 4) Have you ever used variables in programming? (keywords: for, while)
- 5) Have you ever stored different kinds of data in variables in programming?
- 6) Have you ever used variables to store different types of data automatically in programming? (not by hand but as a result of other programming process or algorithm)

Problem #3

Most Difficult Programming Technique: Picture pixel array management

Category: Data display

Hint: pay attention to how the pixels of each sample's picture change!



画像サンプル1



画像サンプル2

How to handle:

- 1) Place the mouse over the left border of any of the samples
- 2) Move the mouse SLOWLY from the left border to the right border of the sample.
- 3) Finally, move the mouse pointer from the right border to the left border and see what happens!

Correct Answer: Sample 2.

Why? In the first sample, the picture is changing completely according to the mouse movement, while in the second one each pixel on the picture's pixel matrix is being replaced by a circle that changes its size according to mouse movement.

Questions:

- 1) After reading the previous explanation did you understand how “Picture pixel array management” is used in this problem?
- 2) Did you realize about the use of “Picture pixel array management” on this problem before reading the explanation?
- 3) Have you ever applied “filters” (or options to manipulate the pixels of an image) to pictures by using software tools? (Microsoft Paint, Adobe Photoshop, etc.)
- 4) Have you ever made filters for pictures directly with programming? (to change picture properties like saturation, contrast, brightness etc.)

Problem #4

Most Difficult Programming Technique: Recursion

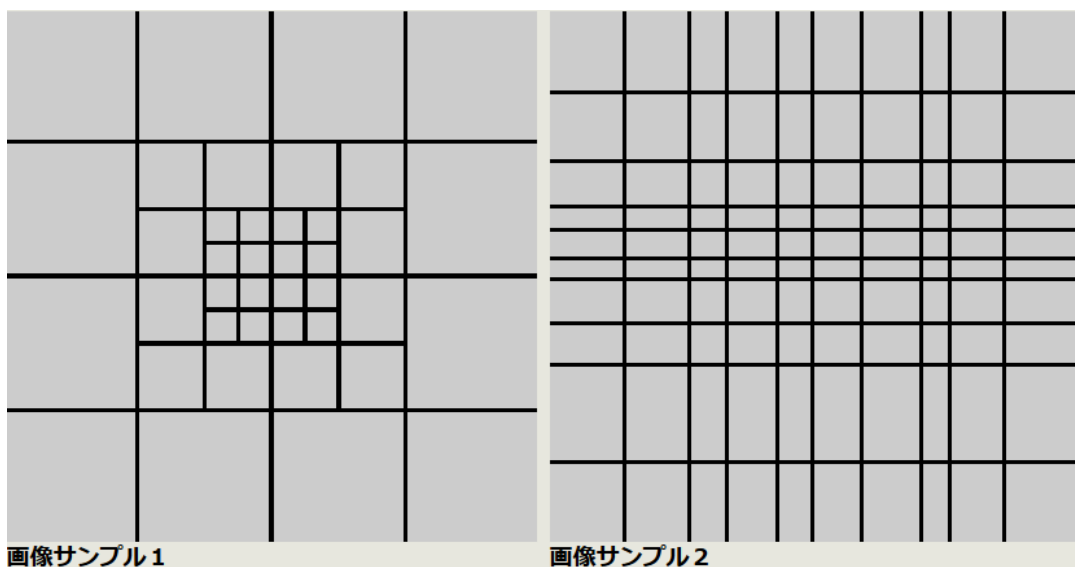
Category: Data Processing

Hint: Look at the length and position of each horizontal and vertical lines

How to handle:

- 1) Place the mouse over any of the samples
- 2) Press the right click button.
- 3) Change the mouse position anywhere inside each sample, press right click again and see what happens!

Correct Answer: Sample 2.



Why? In the first sample, the picture is changing completely according to the mouse movement, while in the second one each pixel on the picture's pixel matrix is being replaced by a circle that changes its size according to mouse movement.

Questions:

- 1) After reading the previous explanation did you understand how “Picture pixel array management” is used in this problem?
- 2) Did you realize about the use of “Picture pixel array management” on this problem before reading the explanation?

- 3) Have you ever applied “filters” (or options to manipulate the pixels of an image) to pictures by using software tools? (Microsoft Paint, Adobe Photoshop, etc.)
- 4) Have you ever made filters for pictures directly with programming? (to change picture properties like saturation, contrast, brightness etc.)

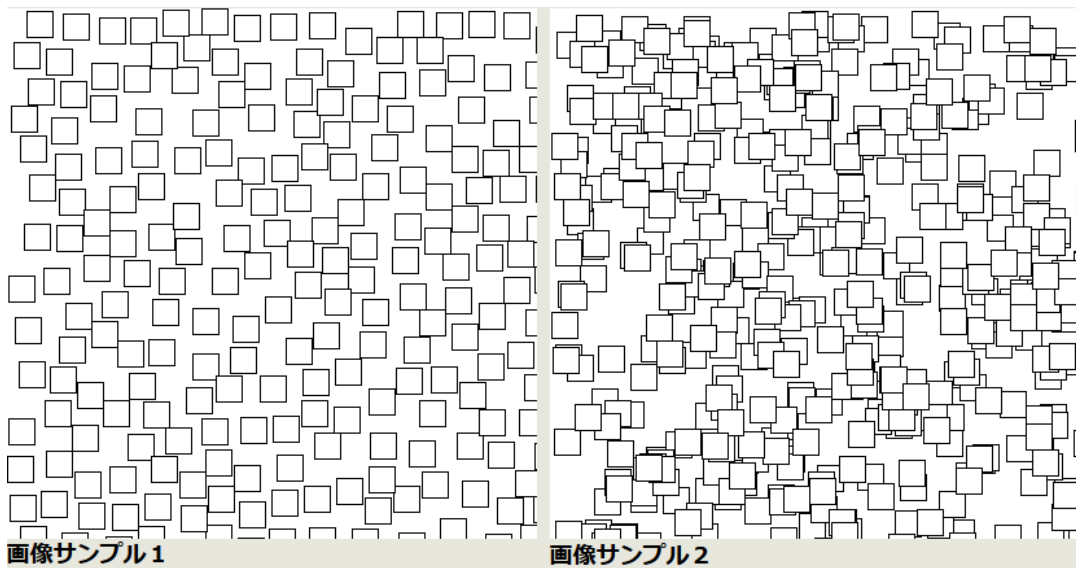
Problem #5

Most Difficult Programming Technique: Empty area detection

Category: Data Display

Hint: Pay attention to the position of the squares on each sample!

How to handle: There is no need to use the mouse here, just look at the animation on each sample and answer the question below. If you need to see the animation again, please press the “see samples again” button beside the submit button.



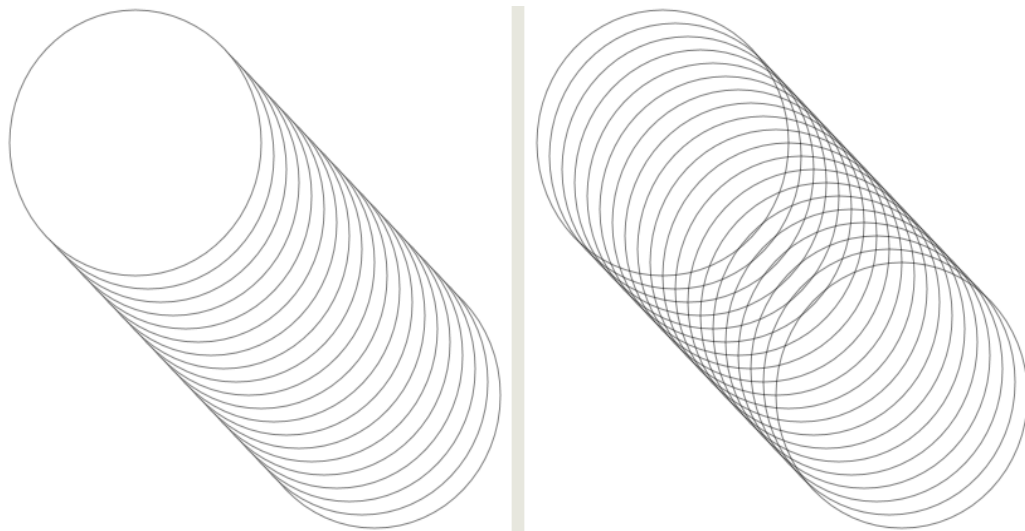
Correct Answer: Sample 1.

Why? While the second sample places squares following a random function without criteria related to where to place them, the first sample is applying an empty space recognizing algorithm that loops throughout all the already placed squares to see if there is an empty space and if this is big enough to contain a new square, therefore the second one is the most difficult sample.

Questions:

- 1) After reading the previous explanation did you understand how “Empty Area Detection” is used in this problem?
- 2) Did you realize about the use of “Empty Area Detection” on this problem before reading the explanation?
- 3) Have you ever used “layers” in software tools? (layers are areas that can contain many objects and can be changed independently of other similar “layers”, you can find them in software like word, PowerPoint, Photoshop and others)
- 4) Have you ever used conditionals in programming? (keywords: if, while)
- 5) Have you ever drawn elements on screen with programming depending on the result of other processes (for example, drawing many rectangles with a loop)?

Problem #6



画像サンプル1

画像サンプル2

Most Difficult Programming Technique: Hidden line removal

Category: Data Display

Hint: How do you get the circles to be transparent? and to be overlapped?

How to handle: There is no need to use the mouse here, just look at the picture on each program sample and answer the question below.

Correct Answer: Sample 1.

Why? The first example even though it looks simple (a repetition of filled circles) it involves the programming concept of "hidden line removal" where lines covered by "surfaces" (color or texture) with borders intersecting are not drawn according to the position of the line or circle immediately above (or at the right in this case).

Questions:

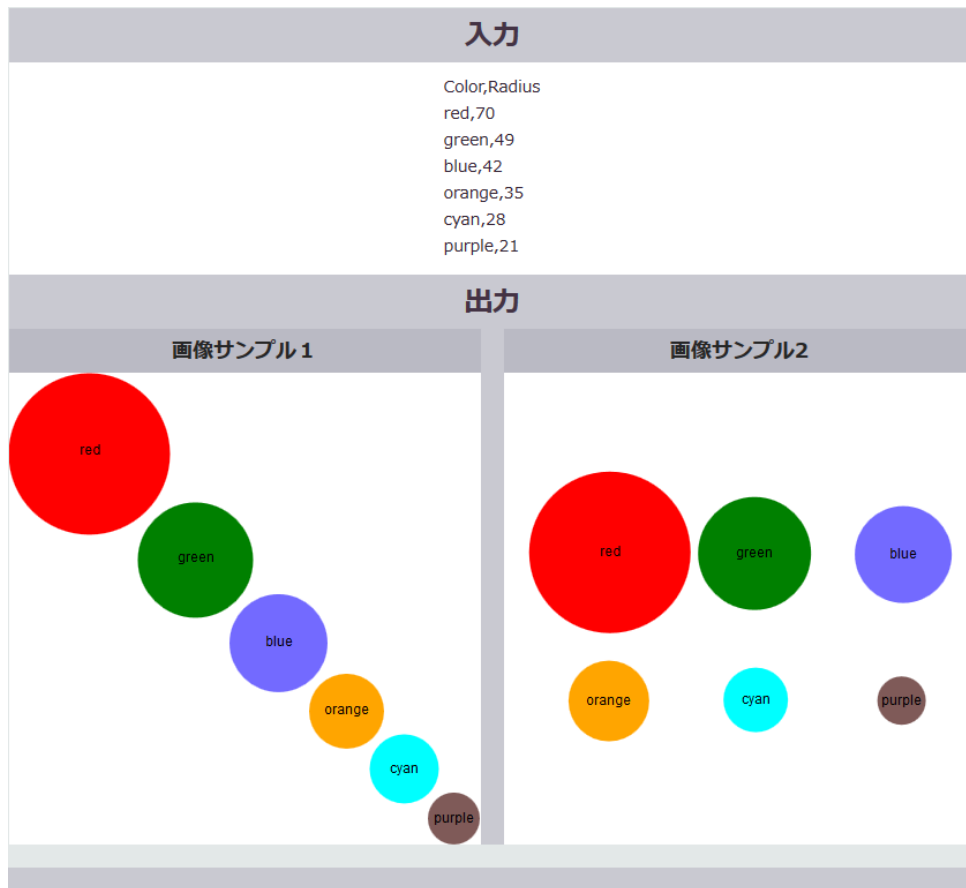
- 1) After reading the previous explanation did you understand how “Hidden line removal” is used in this problem?
- 2) Did you realize about the use of “Hidden line removal” on this problem before reading the explanation?
- 3) Have you ever drawn overlapped figures by using software tools? (like: Word, PowerPoint, Adobe Photoshop, or Adobe Illustrator etc.)
- 4) Have you ever tried to draw a cube with rectangles on software tools? (example: in Microsoft paint, draw two overlapped rectangles and joint them together with lines)
- 5) Have you ever drawn overlapped shapes with programming? (one shape over another, intersecting)
- 6) Have you ever drawn a cube by using 2d figures with programming?

Problem #7

Most Difficult Programming Technique: Distribute Objects According to Data

Category: Data Display

Hint: Think about how the circles are arranged and how you can get these arrangements with programming!



How to handle:

Input data: Is the color and the radius of different circles ordered from the biggest to the smallest.

Output graphs: Circles drawn using the input data color and radius.

Correct Answer: Sample 1.

Why? The most difficult programming technique to identify in the output sample 1 is: to calculate the necessary space between the circles having into account each circle size written in the input data and fit them into the square diagonal; While on sample 2 is to do a grid of points equally separated and position the circles from the biggest to the smallest one, matching the center of each circle with each point, the position or the space between circles don't depend on the size.

Questions:

- 1) After reading the previous explanation did you understand how “Distribute Objects According to Data” is used in this problem?

- 2) Did you realize about the use of “Distribute Objects According to Data” on this problem before reading the explanation?
- 3) Have you ever tried to draw shapes of different sizes distributed with the same space between them by using drawing software tools? (Example: Microsoft Paint, Photoshop etc.)
- 4) Have you ever used the “align and distribute” option that makes objects align to center, left or right and distribute the space between them in software tools? (Example: Microsoft PowerPoint, Photoshop etc.)
- 5) Have you ever changed the properties of shapes (size, color, position etc.) depending on data located in external files with programming?
- 6) Have you ever written code to align and distribute shapes or objects in a pattern (ex: distribute shapes in a circle pattern or in a line)?

Problem #8

Most Difficult Programming Technique: Multiple Data Sources

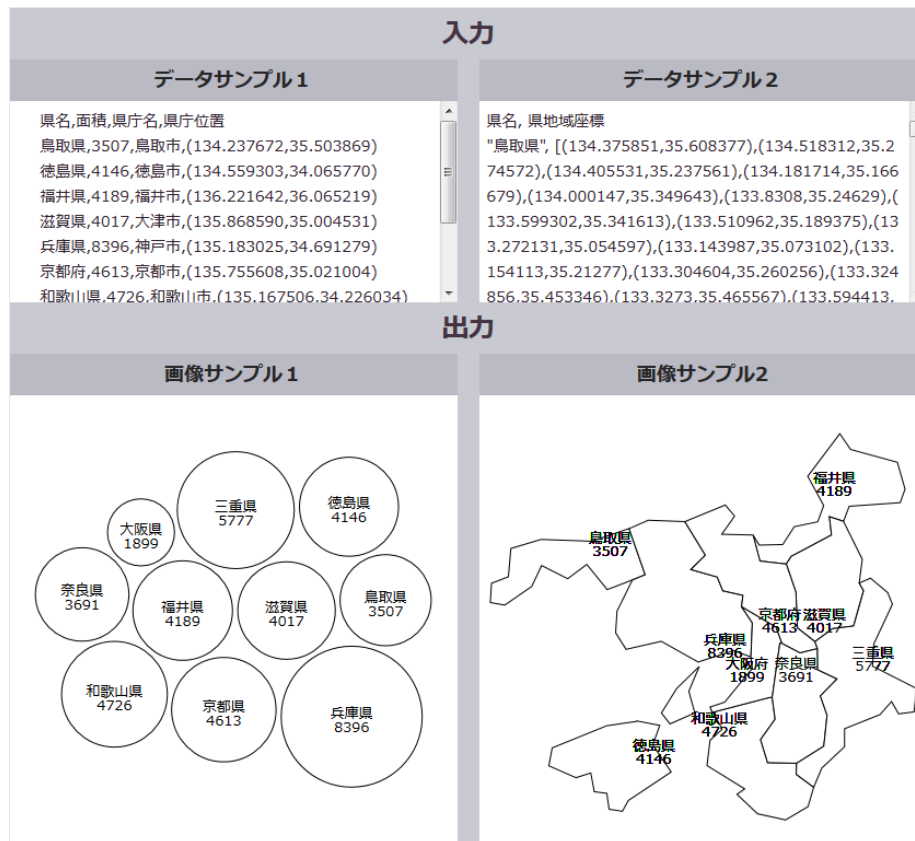
Category: Data Processing

Hint: Pay attention to how the input data is used on each output, and think if there are places where the input data is not used!

How to handle:

Input data: sample 1 contains the name and total area of some Japanese prefectures, besides of the name and coordinate of their main cities. Sample 2 contains the coordinates for initial and end points to draw the limit line of each one of the prefectures on sample 1.

Output graphs: sample 1 is a circle diagram that uses each prefecture area as its size and is ordered by using a “nearest neighbor” algorithm. Sample 2 is a map of the prefectures drawn by using the input data.



Correct Answer: Sample 1.

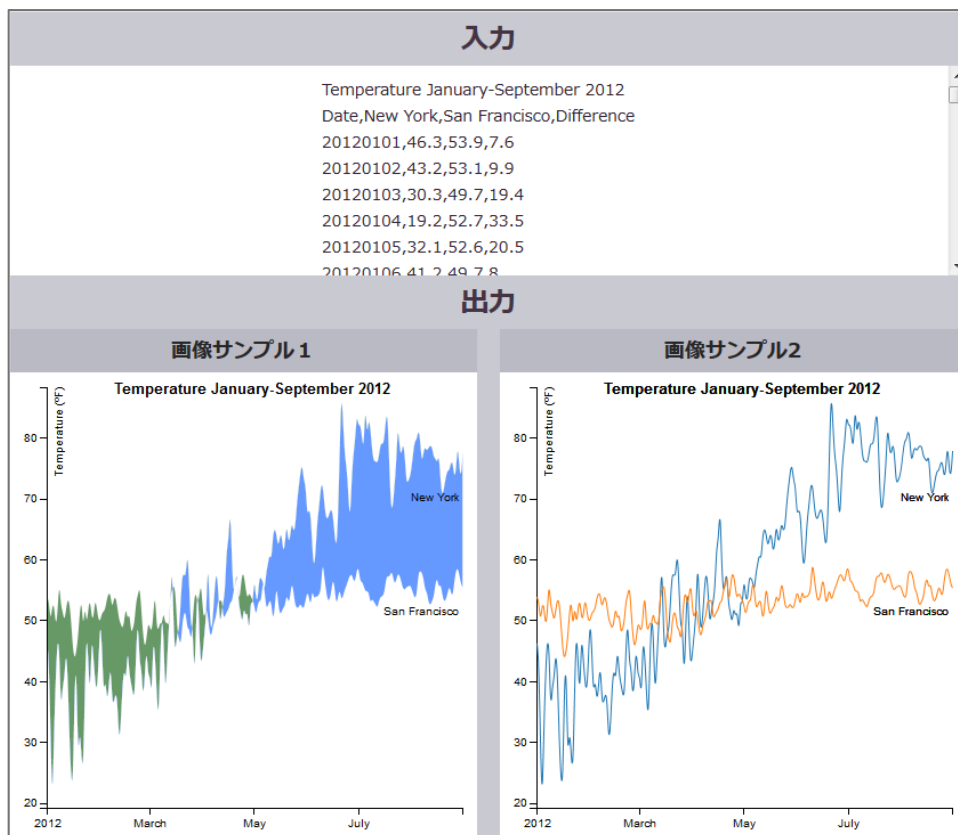
Why? The most difficult programming technique to identify in the output sample 1 is: to find the circle nearest to each other according to the size of each circle (that is the area value of each prefecture according to the first data sample); While on sample 2 to trace lines from one point to another according to the coordinates of each prefecture map provided in the second input data sample, additionally, to use the coordinate of each prefecture’s capital city to position its name and area.

Questions:

- 1) After reading the previous explanation did you understand how “Multiple Data Sources” are used in this problem?
- 2) Did you realize about the use of “Multiple Data Sources” on this problem before reading the explanation?
- 3) Have you ever drawn by using vectors in software tools? (ex: if you draw lines in Word or PowerPoint, you are drawing vectors)?

- 4) Have you ever drawn maps by using software tools? (ex: vector based software tools like: Adobe Illustrator, Corel Draw, or even Microsoft PowerPoint can be used)
- 5) Have you ever used a GPS by inputting a destination coordinate or point? (ex: insert a place coordinate in the GPS of a car)
- 6) Have you ever changed the properties of shapes, or data depending on multiple external data sources in software tools? (ex: in excel you can change or show data on one sheet according to changes on multiple other data sheets or files)?
- 7) Have you ever used coordinates in programming? (ex: X and Y coordinates)
- 8) Have you ever used map coordinates in programming? (ex: similar to the GPS coordinates or similar to the ones on the input data samples)

Problem #9



Most Difficult Programming Technique: Visualization According to Data Analysis

Category: Data Display

Hint: Pay attention to the colored parts of each sample, and where the color difference starts and ends!

How to handle:

Input data: Temperature in two American cities between January and September of 2012, first line are the labels of data, from the second line and each line reports the data per day

Output graphs: Sample 1 and sample 2 are the same line graphs but sample 1 has colored the area between the lines according to which city has the highest temperature on each day.

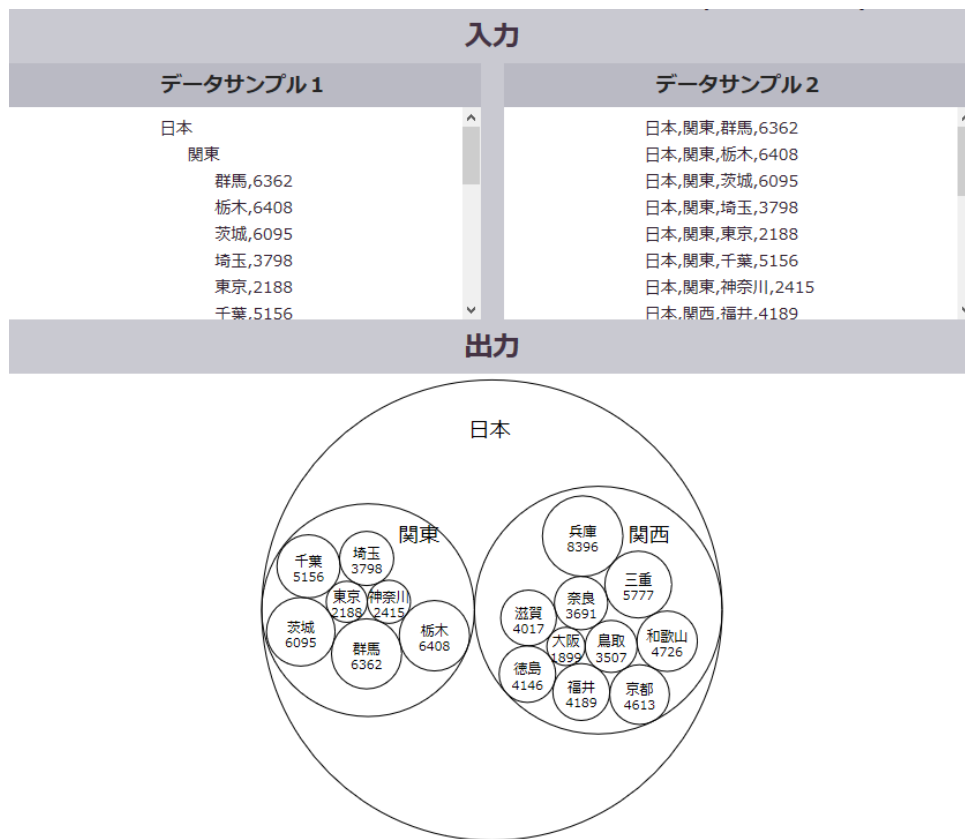
Correct Answer: Sample 1.

Why? Both samples are drawing the same graph from the data but the program of sample 1 has to compare data in both cities each day to know which city is higher, besides having into account those points, calculate the area between both lines and color this area accordingly. To calculate the area between both lines according to the data and then apply color is a difficult process therefore sample 1 is more difficult.

Questions:

- 1) After reading the previous explanation did you understand how “Visualization According to Data Analysis” is used in this problem?
- 2) Did you realize about the use of “Visualization According to Data Analysis” on this problem before reading the explanation?
- 3) Have you ever made polygons with lines by using software tools? (for example, in PowerPoint you can join together lines to build a polygon then you can change its shape from its vertices or apply color inside)
- 4) Have you ever made polygons with lines with programming (doing line by line and make the program consider a group of lines joined together a polygon)?
- 5) Have you ever calculated the area between two curves in mathematics? (calculus)
- 6) Have you ever calculated the area between two curves with programming?

Problem #10



Most Difficult Programming Technique: Reading Data Formats

Category: Data Processing

Hint: Pay attention to how the data samples are written!

How to handle:

Input data: sample 1 and sample 2 contain the name and total area of some Japanese prefectures ordered according the region (kanto or kansai), in sample 1 the lines shifted to the left are parents of (or containers of) the lines shifted to the right, for example: if 日本 is shifted to the left and 関東 is shifted to the right that means that 日本 contains 関東 or 日本 is a parent of 関東. In sample 2 each line contains the name of the country, the region, the prefecture and the area.

Output graphs: is a circle packing diagram of the hierarchy taken from the data input.

Correct Answer: Sample 1.

Why? It is more difficult to do a program to get the output picture with data sample 1 because the program will need to perform additional processes to identify which characters indicate the levels of hierarchy (in this case, spaces), besides, the program will need to confirm for each line who is his parent and children, and finally, it will also need to sort the data.

Questions:

- 1) After reading the previous explanation did you understand how “Reading Data Formats” is used in this problem?
- 2) Did you realize about the use of “Reading Data Formats” on this problem before reading the explanation?
- 3) Have you ever transformed spreadsheets into csv (comma separated value) files? (for example: in excel you can transform a table to a comma separated value file)?
- 4) Have you ever opened text files or a comma separated value (csv) files in text processors or spreadsheet software (ex: Word or Excel)?
- 5) Have you ever done a program to read data from different text formats?

Problem #11

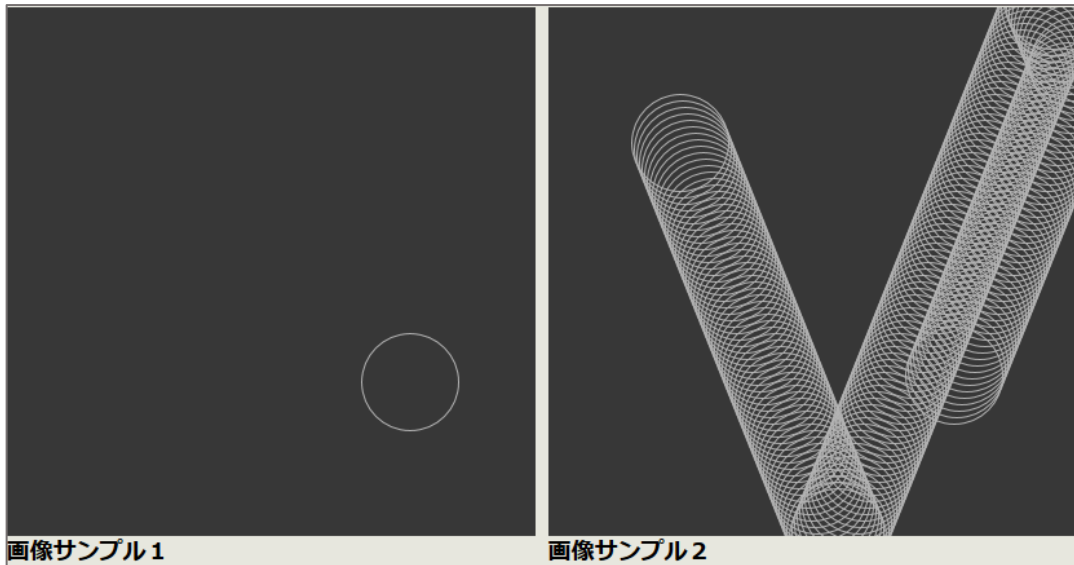
Most Difficult Programming Technique: Erase and Rewrite

Category: Data Processing

Hint: Think about how you can get the circles to move that way!

How to handle: There is no need to use the mouse here, just look at the animation on each sample and answer the question below. If you need to see the animation again, please press the “see samples again” button on the title bar.

Correct Answer: Sample 1.



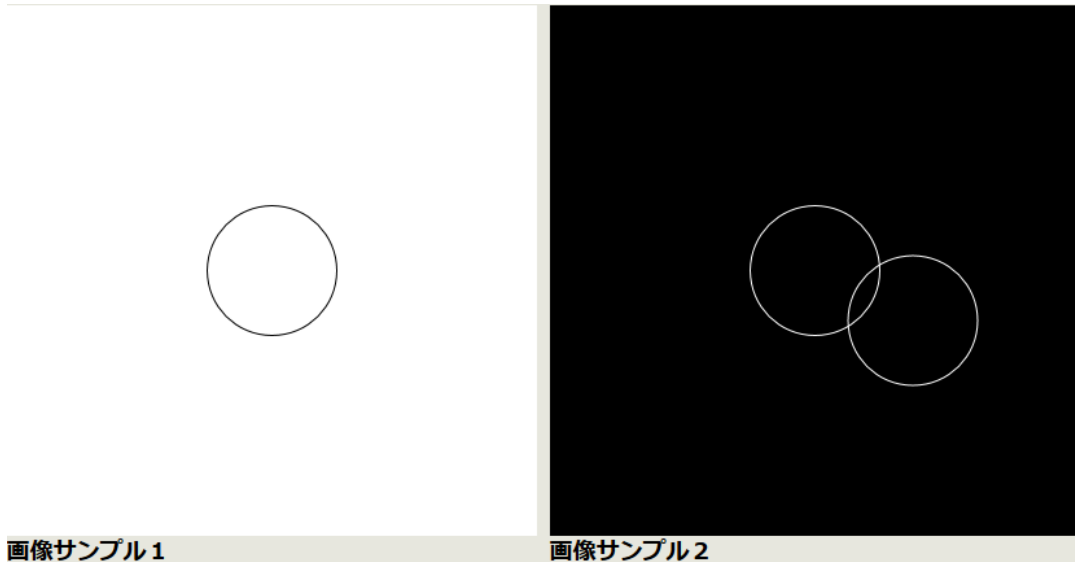
Why? The first sample involves the previous frame calculation. The program is detecting the position of the circle on the previous frame and displaying it, the second sample is not considering the previous frame and is only displaying the current position of the circle.

Questions:

- 1) After reading the previous explanation did you understand how “erase and rewrite” is used in this problem?
- 2) Did you realize about the use of “erase and rewrite” on this problem before reading the explanation?
- 3) Have you ever done small animations by hand or using software tools (Ex: paper flipbooks, animating slides in PowerPoint)?
- 4) Have you ever used commands to erase and rewrite data on software tools? (Example: in operating systems like Windows or Linux there are commands to erase and rewrite data executables from a command line)
- 5) Have you ever erased and rewritten data files with programming? (by using any programming language)?

- 6) Have you ever programmed by yourself code to erase and rewrite data depending on other processes? (Ex: erase and rewrite data on an external file if text appears on screen)

Problem #12



Most Difficult Programming Technique: Invisible events

Category: Data Display

Hint: Pay attention to the mouse position when the color changes.

How to handle:

- 1) Place the mouse over any of the circles located at the center of any of the samples
- 2) Move the mouse outside the circle
- 3) On sample #2 try to make the border of both circles touch and see what happens!

Correct Answer: Sample 1.

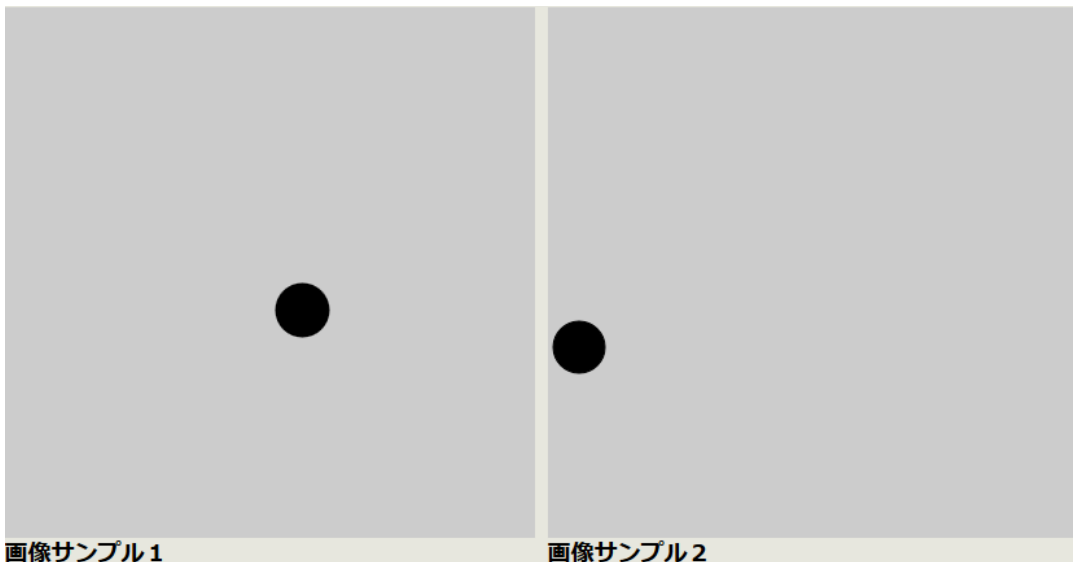
Why? This program is the same for both samples, but in the second sample the mouse pointer is changed to a circle with the same size of the one fixed in the middle of the sample, and the area detected by the mouse is invisible and a little bigger than the circle in the middle.

Questions:

- 1) After reading the previous explanation did you understand how “invisible events” is used in this problem?

- 2) Did you realize about the use of “invisible events” on this problem before reading the explanation?
- 3) Have you ever created clickable pictures? (or pictures that react to mouse input) with any software tool (Ex: in PowerPoint you can make clickable pictures)?
- 4) Have you ever created buttons with programming?
- 5) Have you ever created invisible buttons with programming? (invisible buttons are areas without any picture or drawing, not visible, that react to the mouse when clicked or touched)

Problem #13



Most Difficult Programming Technique: Handling time, speed and distance

Category: Data Display

Hint: Pay attention to the speed of each ball!

How to handle:

- 1) Place the mouse over any of the samples.
- 2) Move the mouse slowly in any direction and see what happens!

Correct Answer: Sample 1.

Why? In this problem time, speed and distance are important to move the balls. In the second sample, the speed of the ball and the mouse is the same so there is no variation on distance and time, while the speed of the ball is different in the first sample, so the time to reach the mouse pointer is different depending on the distance between the pointer and the mouse.

Questions:

- 1) After reading the previous explanation did you understand how “Handling of time, speed and distance” is used in this problem?
- 2) Did you realize about the use of “Handling of time, speed and distance” on this problem before reading the explanation?
- 3) Have you ever moved (or animated) shapes or objects with software tools? (ex: animating shapes or elements in PowerPoint)
- 4) Have you ever changed speed or acceleration of objects in software tools? (ex: fade text or slides, or change speed of presentation slides or shapes in PowerPoint, or change the speed of video with any video editor like: adobe premiere or Sony Vegas)
- 5) Have you ever moved (or animated) shapes or objects with programming?
- 6) Have you ever programmed movement of objects based on a dependence on time, speed and distance? (for example, to move a ball with the same speed)

Problem #14

Most Difficult Programming Technique: Timed action

Category: Data Display

Hint: Think of what could be activating the lines drawing on each sample!

How to handle:

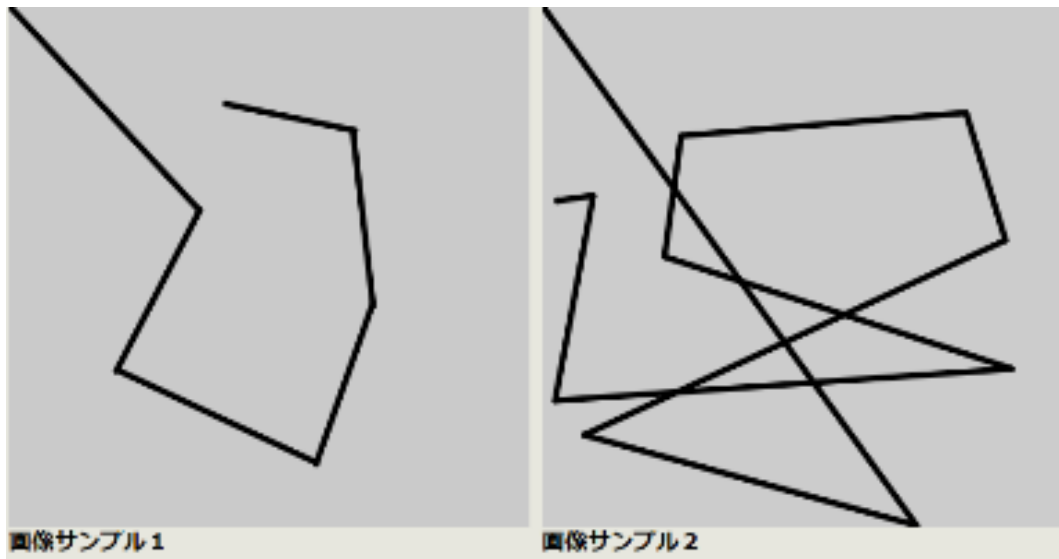
For Sample1

- 1) Place the mouse over sample 1

- 2) Press the right Click
- 3) Change the position of the mouse inside the sample and press the right click again.

For sample 2

- 1) Place the mouse over sample 2
- 2) Change the mouse position inside the sample when a line appears automatically (1 second approx.)



Correct Answer: Sample 2.

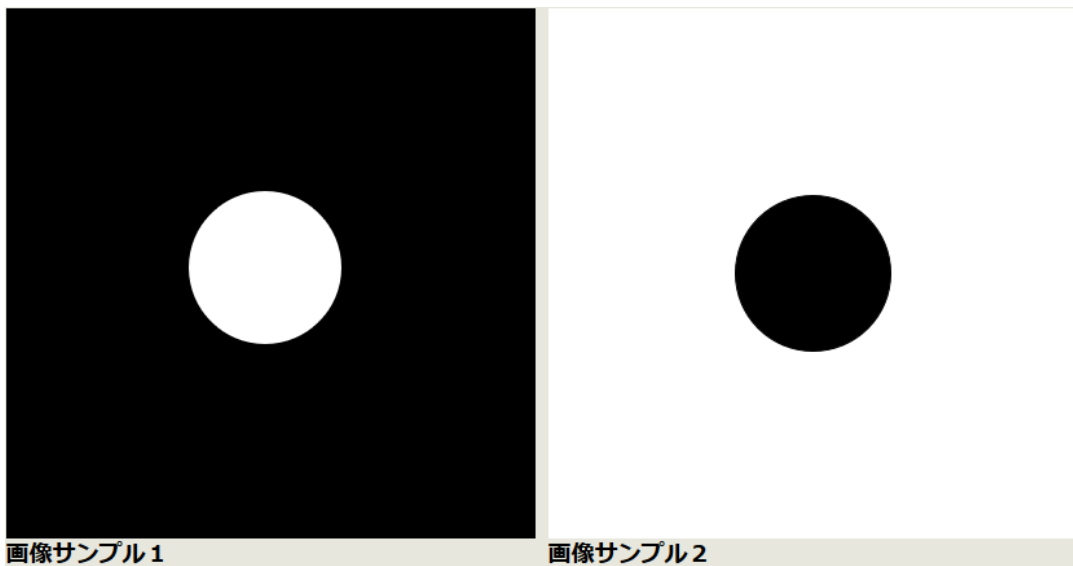
Why? The programming process to draw the lines is the same for both samples but while the lines of the first sample are triggered by the mouse the second sample includes additionally a timer algorithm that triggers the drawing after one second has passed.

Questions:

- 1) After reading the previous explanation did you understand how “Timed action” is used in this problem?
- 2) Did you realize about the use of “Timed action” on this problem before reading the explanation?
- 3) Have you ever used a “timeline” to control video or animations in software tools? (example: The animation pane in PowerPoint is a timeline, similar to the timeline of other video editing software like adobe premiere or flash)

- 4) Have you ever used a time counter with programming? (ex: to count how many seconds have passed)
- 5) Have you ever moved (or animated) shapes or objects with programming?
- 6) Have you ever programmed code to activate actions or events depending on time? (ex: print in screen text after some seconds have passed)

Problem #15



Most Difficult Programming Technique: Clickable area

Category: Data Display

Hint: Pay attention of where you are clicking and what happens!

How to handle:

For Sample1

- 1) Place the mouse over any point of sample 1
- 2) Press the right Click more than 2 times (as many as you want)

For sample 2

- 1) Place the mouse over the circle in the center of sample 2
- 2) Press the right click on that circle.
- 3) Press the right click again on the circle.
- 4) Press right click in the area outside the circle inside the sample and see what happens!

Correct Answer: Sample 2.

Why? In the first sample, wherever the click is made, there is an action executed (color change), but in the second sample there is only one sensible area, the action can be executed only by clicking on the circle in the middle.

Questions:

- 1) After reading the previous explanation did you understand how “Clickable area” is used in this problem?
- 2) Did you realize about the use of “Clickable area” on this problem before reading the explanation?
- 3) Have you ever created clickable elements (not buttons but shapes or drawings that activate actions) with software tools (like PowerPoint, word etc.)?
- 4) Have you ever done an object different to a button (ex: a shape, or text etc.) to trigger an action with programming?

Problem #16



Most Difficult Programming Technique: Random

Category: Data Processing/Data display

Hint: Think on how the lines are moving and what is the pattern they are following!

How to handle:

There is no need to use the mouse here, just look at the programmed animation on each sample and answer the question below.

Correct Answer: Sample 1.

Why? The first sample moves the line using a random function simulating a random walk considering the previous position of the line, the second one is moving always upwards in a linear fashion.

Questions:

- 1) After reading the previous explanation did you understand how “Random” is used in this problem?
- 2) Did you realize about the use of “Random” on this problem before reading the explanation?
- 3) Have you ever animated elements applying random movement options in software tools (example: using “random bars” in PowerPoint)?
- 4) Have you ever used random functions in programming? (any programming language)
- 5) Have you ever programmed code by yourself to generate random numbers?

Problem #17

Most Difficult Programming Technique: Trigonometric animation

Category: Data display

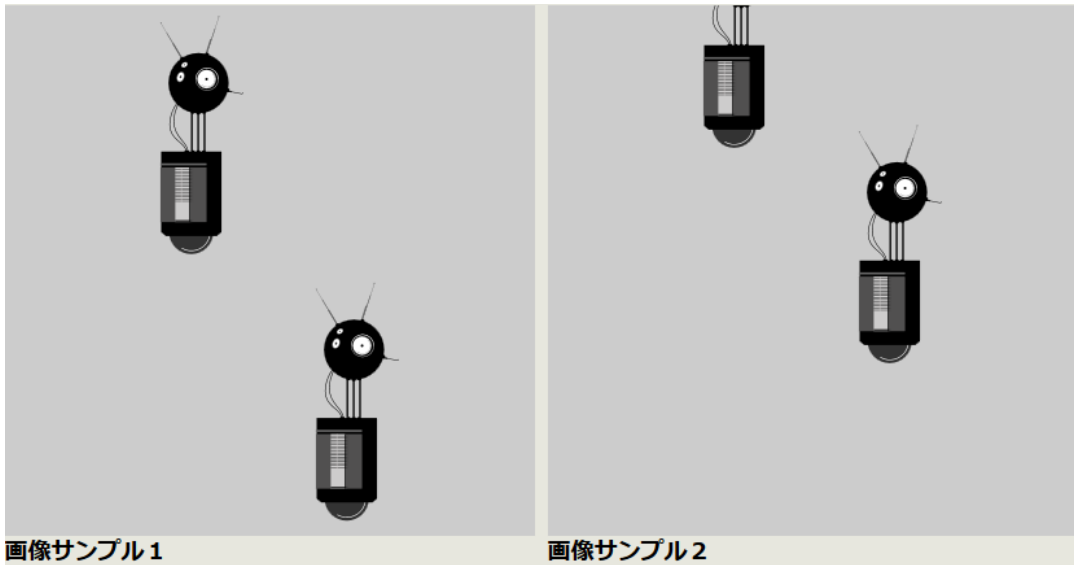
Hint: The way the characters move is showing a mathematical function!

How to handle:

There is no need to use the mouse here, just look at the programmed animation on each sample and answer the question below.

Correct Answer: Both samples have similar difficulty.

Why? Sample 1 moves the character according to a "Sine" trigonometric function, while Sample 2 moves it according to a "Tan" function.



Questions:

- 1) After reading the previous explanation did you understand how “Trigonometric animation” is used in this problem?
- 2) Did you realize about the use of “Trigonometric animation” on this problem before reading the explanation?
- 3) Have you ever done graphs of trigonometric functions on paper (Ex: the basic “sine” function graph)?
- 4) Have you ever animated elements using paths in any software tool (PowerPoint, flash, after-effects etc.)?
- 5) Have you ever animated objects by using trigonometric functions in programming?

Problem #18

Most Difficult Programming Technique: Lists elements inserting

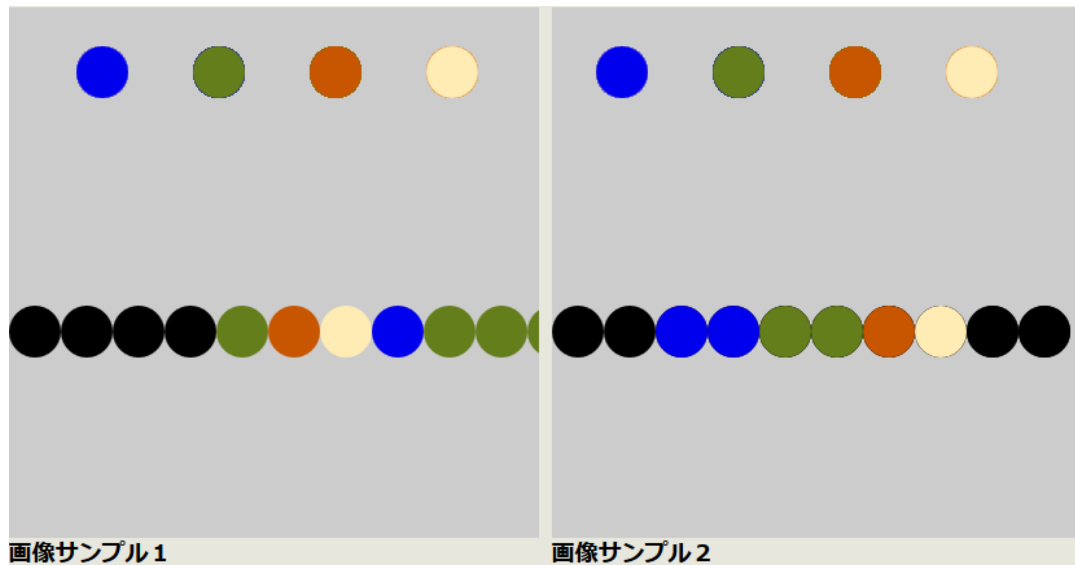
Category: Data Processing

Hint: Pay attention to where the newest ball is inserted with each click!

How to handle:

- 4) Place the mouse over any of the 4 colored circles in any sample.
- 5) Press the right click button.
- 6) Place the mouse on a different colored circle, press right click again and see what happens!

Correct Answer: Sample 2.



Why? Both samples include an array of black balls and 4 clickable colored balls, when any of the colored balls is clicked on the first sample, the clicked colored ball is "appended" to the end of the black ball array; on the other hand, when any colored ball is clicked in the second sample the colored ball is "inserted" or placed between the black balls or between a colored and a black ball.

Questions:

- 1) After reading the previous explanation did you understand how “Lists elements inserting” is used in this problem?
- 2) Did you realize about the use of “Lists elements inserting” on this problem before reading the explanation?
- 3) Have you ever inserted or appended elements in a spreadsheet (for example: in excel by inserting cells, columns or rows)?

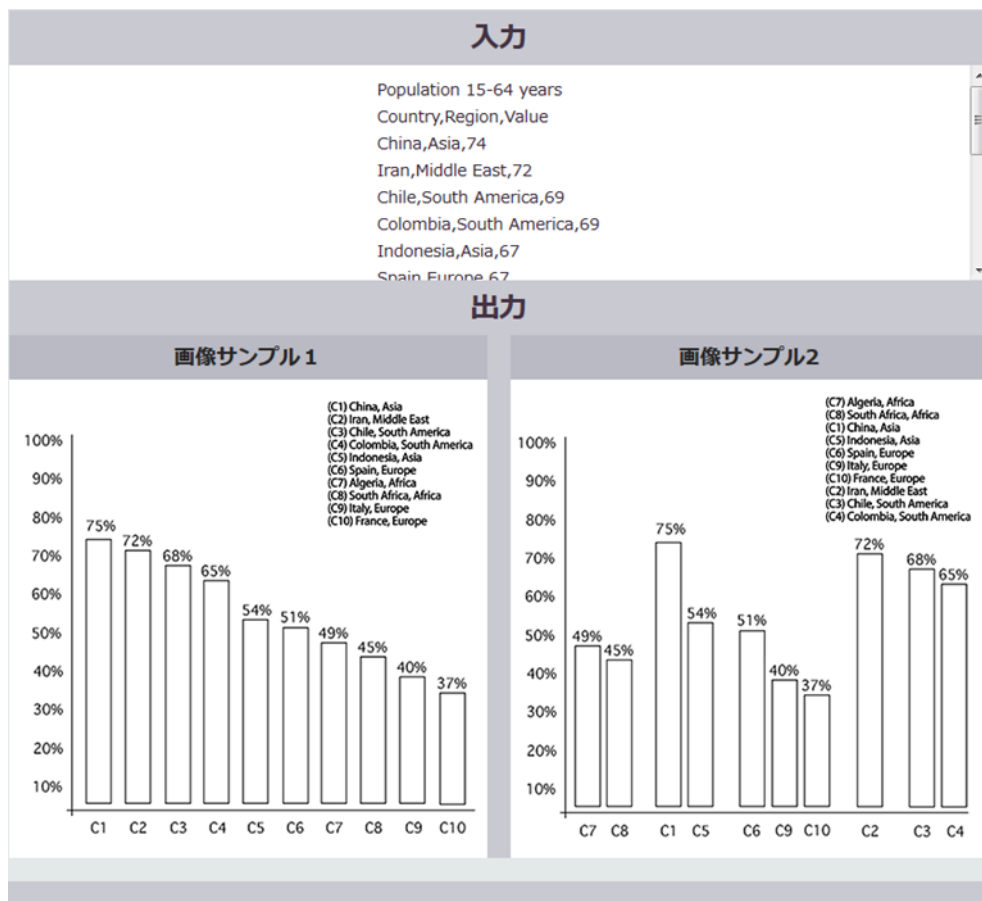
- 4) Have you ever used arrays in programming?
- 5) Have you ever used multidimensional arrays in programming? (example: matrix done with arrays, or an array inside an array)
- 6) Have you ever used ordered lists in programming?

Problem #19

Most Difficult Programming Technique: Mapping data into shapes

Category: Data Display

Hint: Pay attention to the order of the bars!



How to handle:

Input data: Is the population between 15 to 64 years of some countries on different continents ordered from the largest to the smallest, the first line of data is the title, the second data line are the labels, and from there each country data is on one text line.

Output graphs: Bar graphs representing the input data

Correct Answer: Sample 2.

Why? The most difficult programming technique to identify in the graphic output sample 1 is: to map the data into rectangles according to the order given and calculate the proportion according to the data values for each bar (rectangles).

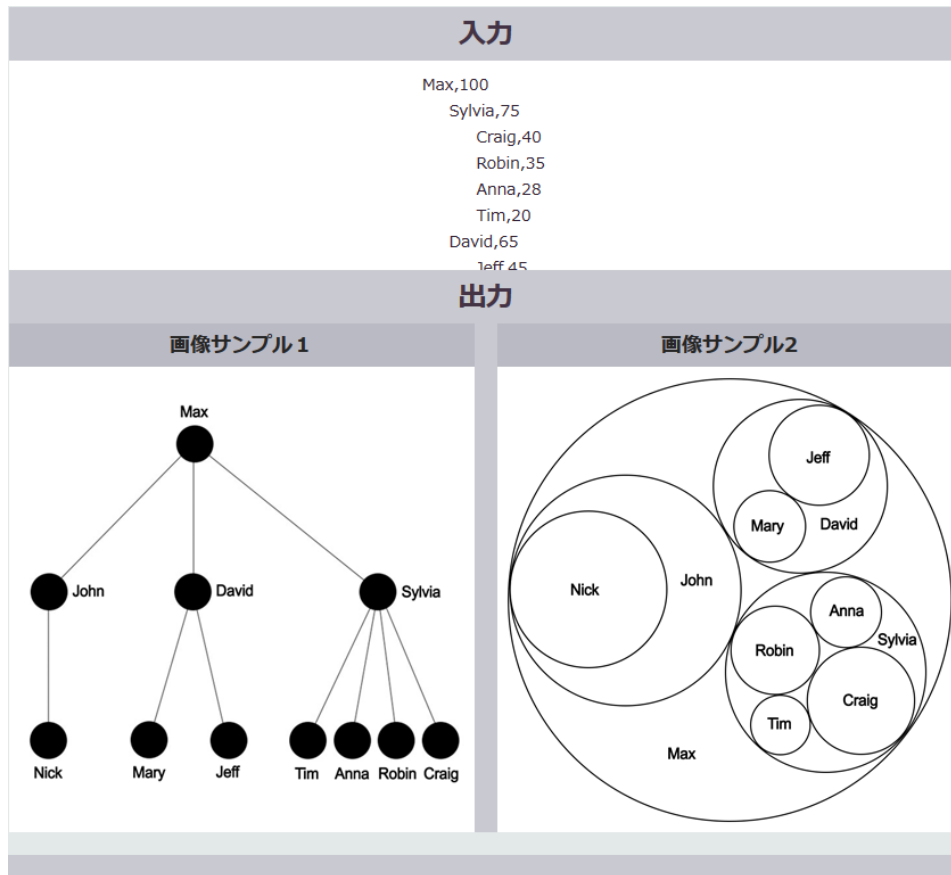
While on sample 2 is to sort the data according to the continent, then inside each continent group, order the data from bigger to smaller, then map each item data population value into bars (rectangles) keeping the proportion and group them together.

Questions:

- 1) After reading the previous explanation did you understand how “Mapping Data into Shapes” is used in this problem?
- 2) Did you realize about the use of “Mapping Data into Shapes” on this problem before reading the explanation?
- 3) Have you ever done a pie graph or a bar graph in software tools? (ex: Word or Excel)
- 4) Have you ever subdivided shapes (a circle into circle arcs or a rectangle into smaller rectangles) with software tools? (ex: PowerPoint, photoshop etc.)
- 5) Have you ever read external data from external sources? (i.e. other file, other computer) by using software tools (ex: A Word or Excel file)?
- 6) Have you ever subdivided shapes with programming? (ex: make arcs of a circle or smaller rectangles inside a rectangle)
- 7) Have you ever made the scale of shapes (size) change according to other programming processes?
- 8) Have you ever read or write data files with programming?

- 9) Have you ever used the data located on external files to change or create shapes or objects with programming?

Problem #20



Most Difficult Programming Technique: Data Hierarchy

Category: Data Processing

Hint: Pay attention at how each sample makes visible the data, and what part of the data is made visible on each sample.!

How to handle:

Input data: Each line is a person and its age, if a line is shifted to the right of the previous line means that this person is a son of the previous one, if they are not shifted or are aligned vertically means that these persons are brothers.

Output graphs: The first sample is a tree map of the hierarchy on the data sample according to the structure of the file. Sample 2 is a circle packing diagram of the hierarchy of the data input, using the age of each person as the size of circles.

Correct Answer: Sample 2.

Why? In the first sample the program does a tree diagram based only on the hierarchy shown on data (namely the lines shifted and the alignment) While in the second sample the program does a circle packing diagram having into account not only the hierarchy but the age (shown as the size of the circles), age determines how much space each circle will take and this depends on the hierarchy too so, the program needs to calculate that.

Questions:

- 1) After reading the previous explanation did you understand how “Data Hierarchy” is used in this problem?
- 2) Did you realize about the use of “Data Hierarchy” on this problem before reading the explanation?
- 3) Have you ever worked with a folders-files like structure in operating systems like Windows?
- 4) Have you ever created organizational charts or concept maps? (example: in programs like Word or PowerPoint there are options to make these organizational charts)
- 5) Have you ever worked with languages to do web pages? (like HTML or XML)?
- 6) Have you worked with probability tree diagrams (in mathematics)?
- 7) Have you ever programmed an object-oriented application (in languages like Java or C++, or web languages like JavaScript or Python)?

Problem #21

Most Difficult Programming Technique: Mapping Data: Scaling and Measuring

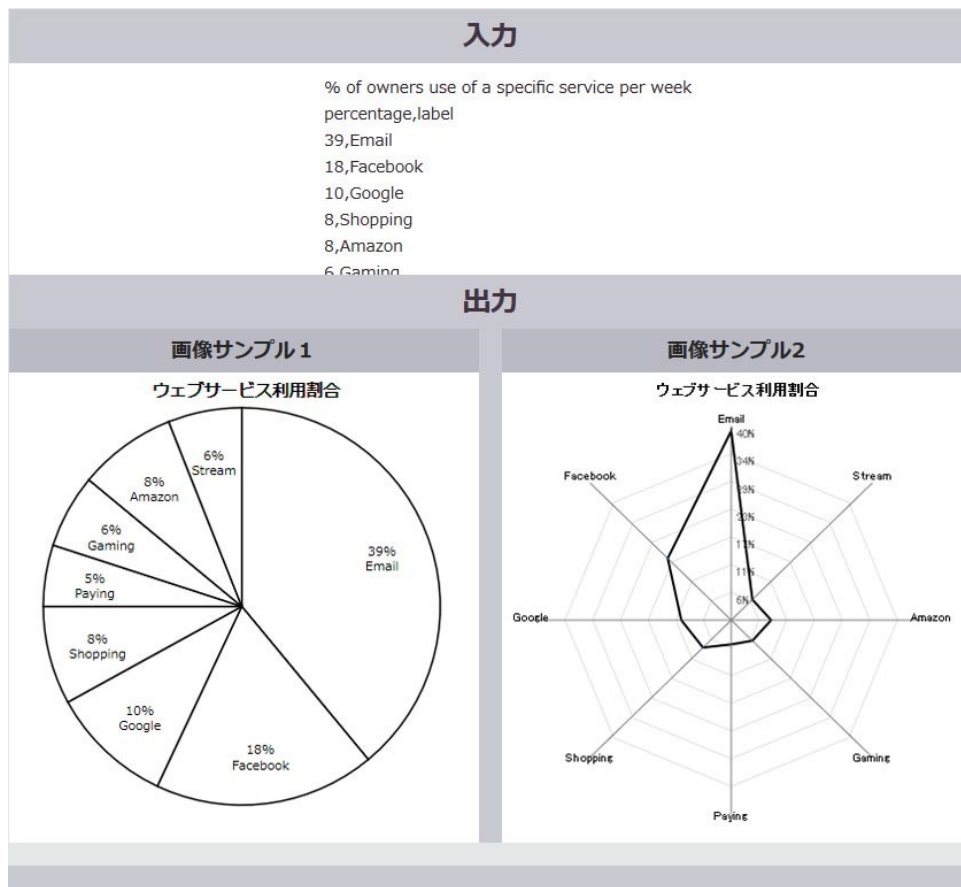
Category: Data Display

Hint: Pay attention to how the data is represented on each sample, and ask yourself what would the program do to make the graphs for the data!

How to handle:

Input data: Percentage of use of a web service per week.

Output graphs: sample 1 is a pie graph and sample 2 is a radial chart, both represent the input data.



Correct Answer: Sample 1.

Why? In order to do both charts the program will need to divide a circle into parts, the difference is on the proportionality of the parts and the additional processes the program will need to do to map the data on the pie chart of sample 1.

For sample 1, the program will need to assign a proportion to each data value on the data sample, that includes divide the circle in as many parts as the data indicates, and then make the arcs (circles divisions) bigger or smaller according to the percentages, in sample 2 the circle has to be divided into the same number of parts but the program then has to divide each part into percentage levels and then trace lines from one percentage point to another according to what data says.

Questions:

- 1) After reading the previous explanation did you understand how “Mapping Data: Scaling and Measuring” is used in this problem?
- 2) Did you realize about the use of “Mapping Data: Scaling and Measuring” on this problem before reading the explanation?
- 3) Have you ever subdivided shapes (a circle into circle arcs or a rectangle into smaller rectangles) with software tools? (ex: PowerPoint, photoshop etc.)
- 4) Have you ever subdivided shapes with programming? (ex: make arcs of a circle or smaller rectangles inside a rectangle)
- 5) Have you ever transformed data from one range to another in mathematics? (example, make a range from 1 to 70 into a percentage range from 1 to 100%)
- 6) Have you ever mapped data (or transformed data from one scale to another) in programming? (any programming language).

Articles Product of this Research

Refereed Articles

1. D. Martinez Calderon, K. Man, H. Kiyomitsu, K. Ohtsuki, Y. Miyamoto and Y. Sun, “An Evaluation Method for Panoramic Understanding of Programming by Comparison with Visual Examples,” *Proceedings of the 2015 IEEE Frontiers in Education Conference (FIE)*, El Paso, Texas, 2015. (The content of this article was presented in chapter 4 of the thesis).
2. D. Martinez Calderon, K. Man, Y. Miyamoto, Y. Sun, M. Hirabayashi, H. Kiyomitsu and K. Ohtsuki, “Measurement Range Increment in a Method for Evaluating Panoramic Understanding of Programming,” *Proceedings of the 2016 IEEE Frontiers in Education Conference*, Erie, Pennsylvania, 2016. (The content of this article was presented in chapter 5 of the thesis).

Non Refereed Articles

1. D. Martinez Calderon, Y. Miyamoto, H. Kiyomitsu and K. Ohtsuki, “Characteristics and Advantages of a Visual Contents Comparison Method for Evaluating Programming Abilities (Article in Japanese)” *Proceedings of 2016 Annual Conference of the Institute of Electrical Engineers of Japan C*, 2016, pp. 1317-1319. (D. Martinez Calderon, 宮本 行庸, 清光 英成, 大月 一弘, “視覚コンテンツ比較によるプログラミング能力評価法の特徴と利点,” 【C】平成 28 年電気学会電子・情報・システム部門大会講演論文集, 2016, pp. 1317-1319.) (The content of this article was presented in chapter 3 of the thesis)
2. D. Martinez Calderon, Y. Miyamoto, H. Kiyomitsu and K. Ohtsuki, “Evaluating Programming Ability by Using a Visual Contents Comparison Method,” *Proceedings of the 9th Data Engineering and Information Management Forum (DEIM2017) (第9回データ工学と情報マネジメントに関するフォーラム(DEIM2017)論文集)*, 2017, pp. 1-4. (The content of this article was presented in chapter 3 of the thesis)
3. D. Martinez Calderon, Y. Miyamoto, M. Hirabayashi, H. Kiyomitsu and K. Ohtsuki, “An Evaluation Method for Panoramic Understanding of Programming by Comparison of Programmed Visual Samples,” *Information Processing Society of Japan, Computers and Education Research Report (研究報告コンピューターと教育(CE))*, Vols. 2016-CE-134, no. 6, 2016, pp. 1-7. (The content of this article was presented in chapter 5 of the thesis)