Effectiveness of Flowcharting as a Scaffolding Tool to Learn Python

Candido Cabo
Departments of Computer Systems
New York City College of Technology/CUNY
New York City, USA
ccabo@citytech.cuny.edu

Abstract—**This Research to Practice Full Paper evaluates the effectiveness of flowcharting as a scaffolding tool to learn a programming language like Python in the setting of an urban institution that serves mostly underrepresented minority students. We found that the abilities of students to solve problems using flowcharts is a good predictor of their ability to solve problems with Python (r-squared = 0.68). This means that the majority of students who perform well using flowcharts will perform well in Python. A majority of students found flowcharting easier than Python (63%), and reported that flowcharting helped them understand how to write programs in Python (73%). However, flowcharting is not a magic bullet for learning programming because about 31% of students have difficulty solving problems with a flowcharting tool (and Python). We also found that the ability of students to read code is not highly correlated with their ability to write code in Python. In conclusion: 1) For a majority of students flowcharting is an effective scaffolding tool to learn Python; 2) The ability to read and trace code is not predictive of the ability of students to solve problems and write viable programs in Python.**

Keywords— Flowcharting, Python, program comprehension, program generation, novice programmers, computer science education;

I. INTRODUCTION

Writing viable computer programs involves many different skills including the ability to understand a word problem, being able to design a solution using programming constructs, and being able to translate the solution into actual code. Students without a previous computer programming experience find all those steps challenging which results in low passing rates in first-year computer programming courses. The challenges faced by novice programming learners have been reviewed in detail elsewhere [1].

In general novice programmers find the syntax and organization of programming languages difficult to learn and understand, which in turn may hinder their ability to design a program that solves a problem. Therefore a common approach to help students ease their way into programming is to separate the design of a solution to a problem (i.e. problem-solving) from the actual code writing [2], [3], [4]. Graphical languages and tools (like flowcharts) have been proposed to represent the computer processes needed to solve problems (and therefore getting away from programming language syntax) [5], [6].

A lack of understanding of the problem domain may also hinder students' ability to find a solution to a problem and write viable computer programs [7], [8], [9], [10]. For example, heavy use of mathematical or accounting problems with students who are not well-prepared in mathematics may be disengaging and become an obstacle to the development of computer programming skills. We have shown that the use of domains that students understand well and feel comfortable with (like videogames and narratives) increases student success in first-year computer programming courses [11], [12].

In light of those earlier findings in computer science education, the first-year programming course at our institution incorporates both computer problem-solving with a flowchart interpreter and with Python. The purpose of this study was to quantify the effectiveness of flowcharting as a scaffolding tool to learn a programming language like Python in an urban institution that serves mostly underrepresented minority students. We compared the effectiveness of flowcharting and Python to develop students' computer problem-solving skills. We also quantified how the ability to read code (i.e. syntax understanding) in Python relates to the ability to write code and solve problems with Python (i.e. being able to design a solution using programming constructs).

The remainder of the paper is organized as follows. Section II provides an overview of prior work. Section III summarizes our research questions. Section IV describes our methodology. Section V shows our empirical results. We finally discuss in Section VI our results in the context of published reports and finally we extract pedagogical conclusions that could be applied in the classroom.

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

II. Background and Prior Work

2.1. Effectiveness of Flowcharts as models and representations of a computer program.

To improve teaching and learning in first year computer programing courses, Soloway and Spohrer [5] suggest the use of graphical languages to illustrate the flow of execution of a program. The evaluation of the effectiveness of graphical languages and tools for teaching programming has been an active area of research in computer science education [13]. Since Von Neumann and collaborators first propose the use of flowcharts to represent computer processes [14], flowcharts have been the most commonly used visual tool available to programmers [13]. Flowcharts provide a reasonable model for procedural computer programming and are good for representing processes, sequences, selection structures and repetition loops. Several empirical studies have tested the value of flowcharts to analyze and understand algorithms. Scanlan [15] compared the use of flowcharts and pseudocode (in programs using sequence and selection structures, but not iteration) and found that, at different levels of algorithmic complexity, flowcharts facilitated and accelerated the comprehension of computer code. Similar results were reported by Vessey and Weber [16] comparing decision trees, pseudocode and decision tables in problems involving conditional logic, and by Cunniff and Taylor [17] when comparing Pascal with a flowchart-based visual programming tool. The overall conclusion of these studies is that visual tools and notations (including flowcharts) provide a better model to represent computer processes than textual representations including computer code, and that the benefit of the visual representation grows with the complexity of the problem [13].

However other studies found that visual notations, including flowcharts, may not be especially helpful in the programming process. Shneiderman et al. [18] found no benefit for novice programmers in tasks involving the comprehension, composition, debugging and modification of computer programs between a group using a flowchart (Fortran + flowchart) and a group not using a flowchart (only Fortran). It is noteworthy that in a later review of this work, Scanlan [15] and Whitley [13] discussed possible design flaws in the Shneiderman et al. [18] study.

Ramsey et al. [19] compared a program design language (a method for designing and documenting software which is related to pseudocode) with flowcharts and found no benefits in the use of flowcharts. Similarly, in another study evaluating experimentally the effectiveness of software documentation formats for presenting information about computer programs, Curtis et al. [20] found that constrained languages (similar to program design languages) were more effective than flowcharts. More importantly, Curtis et al. [20] also reported than more than any specific type of documentation, individual differences between the participants accounted for between one third and one half of the variation in performance. So, subject differences may be more important than differences between tools in computer programming. This may also apply to novice programmers learning a computer language.

It is also worth noticing that while flowcharts are reasonably good representations of computer processes involving sequencing, selection and repetition (i.e. for procedural programming), flowcharts are not suitable to model object-oriented programs. UML diagrams and entity-relationship diagrams are more commonly used to model object-oriented programs than flowcharts. Therefore the benefits of flowcharts may also depend on the problems at hand.

2.2. Comprehension (Reading) vs. Generation (Writing) of Computer Programs.

Both code comprehension and code generation tasks are typically used in the classroom to teach and evaluate students' computer programming skills and competencies. While it is often assumed that there is a high correlation between the abilities to read and write code, reports in the literature point to a more complex relationship. Some studies have shown that there is little correspondence between the ability to read and write code [21], while other studies have shown a strong correlation [22], [23].

Knowledge of syntax and other programming language features is probably the major determinant of the ability to read computer programs. However, Robins et al. [1] argues that the main difficulty in writing viable computer programs is the lack of skills in basic programming design rather than just knowledge of language syntax and features. A number of studies have shown that novice programmers may know the syntax and semantics of individual statements in a programming language, but they may not know how to combine those statements into viable programs [1], [21]. It has been suggested that the teaching of computer programming should focus not only on language syntax but also on the combination and use of those features to solve problems [1], [5]. Understanding the relationship between students' abilities in code reading and writing may have implications on how we teach and evaluate novice programmers.

III. Research Questions

The purpose of this study is two-fold: 1) to quantify the relationship between the students' ability to solve computational problems with a flowcharting tool and with a programming language like Python; 2) to characterize the relationship between being able to read code and to write

effective computer programs in Python. The specific research questions are:

(RQ1) Do students problem-solving abilities differ using a flowcharting tool from using a programming language like Python?

(RQ2) Is flowcharting helpful as a scaffolding tool to learn a programming language like Python?

(RQ3) Which are the students' perceptions about learning flowcharting, programming languages and their relationship?

(RQ4) Does the ability to read code relate to an ability to write effective computer programs?

IV. METHODS

4.1. Participants and setting

Our institution is one of the most racially, ethnically, and culturally diverse institutions of higher education in the northeast United States: 30% of our students are African American, 33% are Latino, 21% are Asian or Pacific Islanders, and 11% are Caucasian. The College's fall 2017 enrollment was 17,279.

All students enrolled in the Computer Systems Technology program at our institution are required to take an introductory Problem-Solving/Computer Programming course. This course is designed to introduce students to concepts of problem solving and computer programming languages. During the first two weeks of the fifteen-week semester, the emphasis is on solving problems in a context (domain) known to the students—for example, navigation of mazes or games such as tic-tac-toe. Computer programming structures that control the flow of execution such as sequencing, selection, and repetition loops are introduced to solve various problems using pseudocode. In the following weeks, students apply those same programming structures to solve problems with flowchart interpreters and Python.

A total of 73 students enrolled in one section of the course in Spring 2017 and in two sections of the course in Fall 2017 were part of the study. Data from students who dropped the class or stopped attending the class (a total of 19 students) were excluded from the analysis.

4.2. Student performance in flowcharting and Python

We measured student performance in problem-solving and computer programming using a flowchart interpreter which allows the execution of the flowchart (www.visuallogic.com) and Python. The problems used to assess the students skills in problem-solving were categorized in three groups: problems using only a sequence of computational steps (sequence), problems that needed the use of selection/decision structures (if/else), and problems requiring the use of repetition loops (both for and while loops).

To evaluate students abilities in code comprehension (i.e. code reading) we used multiple choice assessments in which students were presented with python code and they had to decide whether there was any error in the code presented, and if there was not, which would be the output produced. To evaluate students abilities in code generation (i.e. code writing) we evaluated their ability to write viable programs in Python to solve specific word problems.

4.3. Student survey

At the end of each semester we run a student survey to gauge students' perceptions on learning flowcharting, Python coding and their relationships. The survey contained seven questions and, for each question, students selected one of the following responses: strongly agree, agree, disagree or strongly disagree (see below Figure 3).

V. RESULTS

5.1. Problem-solving with a Flowcharting Tool vs. Problem-solving with Python.

Figure 1 shows student problem-solving performance (scale 0-10) using flowcharts vs. performance using Python. The solid line indicates the linear regression fit of the data in the graph. The dashed lines indicate acceptable performance (70%) and each dot indicates the performances of a given student in flowcharting and Python (note that there may be fewer dots than students because several students may have obtained identical scores in flowcharting and Python). Also shown in the figures are the percent of students in each of the quadrants determined by the 70% performance lines: students performing adequately in both flowcharting and python (upper right quadrant), students performing poorly in both flowcharting and python (lower left quadrant), and students performing acceptably in one but not the other (upper left and lower right quadrants).

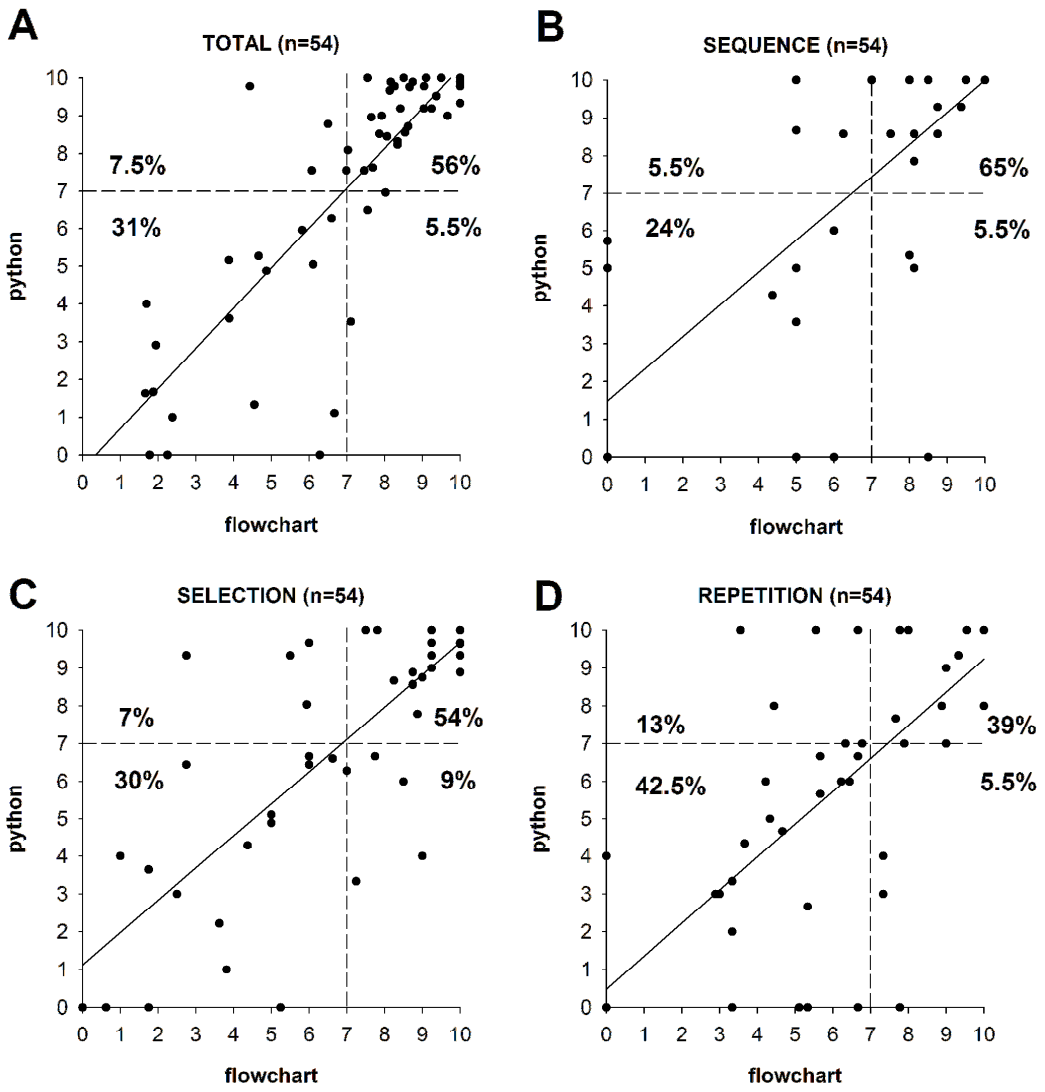

Figure 1. Performance (scale 0-10) in problem-solving with flowcharts vs. code writing in Python. Each student is represented by a dot. Note that there may be fewer dots than students if two or more students have the same performance. The solid line is a regression line. The dashed lines show an acceptable level of performance (70%). The percentages indicate the percent of students in each of the quadrants defined by the dashed lines. (A) Average performances in the three categories of sequencing, selection and repetition (r-squared = 0.68); (B) Performance in problems involving sequencing (r-squared = 0.47); (C) Performance in problems involving selection (r-squared = 0.63); (D) Performance in problems involving repetition (r-squared = 0.49);

Figure 1A summarizes the performance for the average of the three problem categories (sequence, selection, repetition). About 56% of students do acceptably in both flowcharting and Python, and about 31% do poorly in both. A minority of students (5.5%) does well in flowcharting but is not able to transition to Python. On the other hand, about 7.5% do well in Python but not in flowcharting. Of the students who had an adequate understanding of flowcharting, 92% were able to write viable computer programs in Python. Of the students who did not have an adequate understanding of flowcharting only 19% were able to write viable programs in Python. Performance in flowcharting correlates well (i.e. it is a good predictor) with performance in Python (r-squared = 0.68). Still, it should be noted that about one third of the students (31%) do not perform adequately neither in flowcharting nor in Python.

Figures 1B-1D show performance data for each problem category in the same format as for the average (Figure 1A): sequencing (Figure 1B), selection (Figure 1C) and repetition (Figure 1D). The percent of students performing adequately in flowcharting and Python is 65% for problems that only require the use of a sequence of computational steps (Figure 1B), decreases to 54% for problems that require selection (if/else)

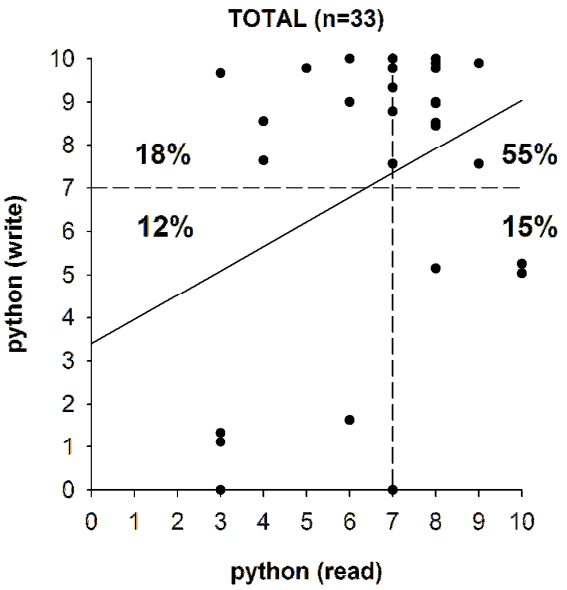

Figure 2. Performance (scale 0-10) in code reading vs. performance in code writing in Python. Each student is represented by a dot. Note that there may be fewer dots than students if two or more students have the same performance. The solid line is a regression line (r-square = 0.10). The dashed lines show an acceptable level of performance (70%). The percentages indicate the percent of students in each of the quadrants defined by the dashed lines.

(Figure 1C), and further decreases to 39% for problems that require repetition loops (Figure 1D).

Still, performance in flowcharting in all categories is a reasonably good predictor of performance in Python: only between 5%-9% of students do well in flowcharting but do not do well in Python. It should also be noted that between 5%-13% of students do well in Python but do not do well in flowcharting.

5.2. Code Comprehension (reading) vs. Code Generation (writing) in Python

Figure 2 (with the same format as Figure 1) illustrates the ability of students to read vs. to write code in Python (33 students enrolled in the two Fall 2017 sections participated in this part of the study). It would be expected that code comprehension and code generation in Python are closely related. However, Figure 2 shows that code reading is not a good predictor of code writing (r-squared = 0.10). About 22% of students who can read Python code cannot translate that ability into writing viable Python programs. Also about 25% of the students who can write Python programs cannot read Python code (written by others). These results show that there is a complicated relationship between code reading and code writing. Most importantly, evaluating the students ability to read code (for example, in a multiple choice type of assessment) does not evaluate the students' ability to solve problems by writing a Python program.

5.3. Students Perceptions of Learning Flowcharting and Python

Figure 3 shows the result of a student survey to gauge students perceptions on learning flowcharting and Python (n = 52 students completed the survey). A vast majority of students (~80%) strongly agree or agree that they enjoyed learning flowcharting and Python. Also about 73%-80% of students agree or strongly agree that the tools used in the course (flowcharting and Python) helped them become better problem solvers.

About 73% of students strongly agree or agree that flowcharting help them understand how to write programs in Python. When asked to compare flowcharting and Python as problem-solving tools, 71% of students responded that they enjoyed Python more than flowcharting, but 66% responded that they found that flowcharting was easier than Python.

VI. DISCUSSION

6.1. Relationship between flowcharting tool and Python as problem-solving tools (RQ1-RQ3)

We have shown that students' abilities to solve problems using flowcharts is a good predictor of their ability to solve problems with Python (RQ1). This means that the majority of students who perform well using flowcharts will not have problems transitioning to Python. A majority of students (66%) finds flowcharting easier than Python, and that flowcharting help them understand how to write programs in Python (73%) (RQ3). All in all it would seem that for a majority of students flowcharting is an effective scaffolding tool to learn Python (RQ2). However, it is interesting to note that for a majority of students Python is more enjoyable than flowcharting (71%). It is possible that once they are able to make the transition to Python, solving a problem with Python is faster and perhaps simpler than using a flowcharting tool. Still, flowcharting is not a magic bullet because about one third of students have difficulty solving problems with a flowcharting tool (and Python).

Overall our results are consistent with the general consensus that visual representations of computer code and processes are more helpful than plain code in representing computer processes [5], [15], especially for novice programmers. However, it is interesting to note that between 5% and 13% of students do well in Python but unsatisfactorily with

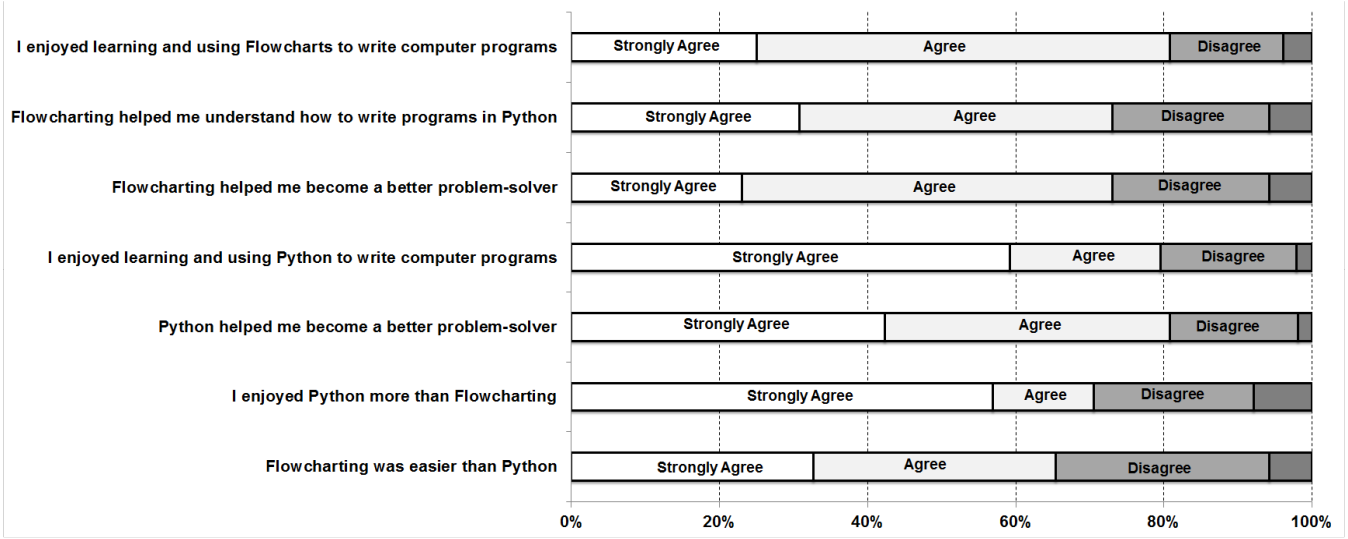

Figure 3. Results of the student survey on learning flowcharting and Python (n = 52). The unlabeled block to the right of "Disagree", in the darkest gray, represents the percent of students who "strongly disagree" with the question.

flowcharts. This may indicate, as suggested by Curtis et al. [20], that performance may not depend on the tool alone, but in the interaction between an individual student and the tool. Personal preferences may determine the level of engagement and consequent performance level using a given tool. Therefore, it is possible that some students are more motivated to learn Python than flowcharting (after all 71% of students find Python more enjoyable), or that they may be more textual than visually oriented. This may explain also why some studies have found that flowcharts may not be helpful in algorithm and program comprehension and/or development [18].

6.2. Relationship between Code reading and code writing (RQ4)

We also found that the ability to read and trace Python code is not predictive of the ability of students to solve problems and write viable programs in Python (RQ4). Therefore, to use code reading prompts to evaluate students' problem-solving abilities is not adequate. Code reading may be a good assessment of their knowledge of Python syntax, but code writing requires additional abilities beyond the syntax of a specific language [1], [5]. Still the relationship between code reading and writing is likely to be complex. While different skills are involved in code reading and writing, it is clear that both code reading and writing are inextricably linked in the process of learning computer programming: reading and copying code is a fundamental step towards learning how to write code. Moreover, the process of getting a computer program to work properly during the debugging process also involves an iterative process of code writing and code reading.

VII. LIMITATIONS

All students included in this study learned and used both flowcharting and Python. It would be interesting to investigate if teaching Python without flowcharting would result in a similar percentage of students who are able to write viable Python programs. Also, it is unknown whether our results can be generalized and can be extrapolated to other student populations and institutions. Further studies will need to be carried out to determine if these results hold in a different context.

VIII. CONCLUSIONS

1) The ability of students to solve problems using flowcharts is a good predictor of their ability to solve problems with Python.
2) A majority of students finds flowcharting easier than Python, and believes that flowcharting helped them understand how to write programs in Python.
3) Therefore, for a majority of students, flowcharting is an effective scaffolding tool to learn Python.
4) The use of code reading prompts to evaluate the abilities of students to solve problems and write code in Python is not adequate.

REFERENCES

[1] Robins A, Rountree J, and Rountree N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education 13: 137-172.

[2] Deek F. P., Kimmel, H., & McHugh, J. A. (1998). Pedagogical changes in the delivery of the first-course in computer science: Problem solving, then programming. *Journal of Engineering Education*, *87*(3), 313-320

[3] Fincher, S. (1999). What are we doing when we teach programming? *Proceedings of IEEE Frontiers in Education Conference* (pp. 12A4/1-12A4/5).

[4] Kay, J., Barg, M., Fekete, A., Greening, T., Hollands, O., Kingston, J., & Crawford, K. (2000). Problem-based learning for foundation computer science courses. *Computer Science Education*, *10*(2), 109-128.

[5] Soloway, E., & Spohrer, J. C. (Eds.) (1989). *Studying the novice programmer*. Hillsdale. NJ: Lawrence Erlbaum.

[6] McSporran, M. and King, C. (2005). Blended Is Better: Choosing Educational Delivery Methods. *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, Montreal, Canada.

[7] Brooks, R. E. (1977). Towards a theory of the cognitive processes in computer programming. *International Journal of Man-Machine Studies*, *9*, 737-751.

[8] Davies, S. P. (1993). Models and theories of programming strategy. International Journal of Man-Machine Studies, 39(2), 237-267

[9] Rist, R. S. (1995). Program structure and design. *Cognitive Science*, *19*, 507-562.

[10] Spohrer, J. C., Soloway, E., & Pope, E. (1989). A goal/plan analysis of buggy Pascal programs. In E. Soloway and J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 355-399). Hillsdale, NJ: Lawrence Erlbaum.

[11] Cabo, C., & Lansiquot, R. D**.** (2013). Development of interdisciplinary problem-solving strategies through games and computer simulations. In R. D. Lansiquot (Ed.), *Cases on Interdisciplinary Research Trends in Science, Technology, Engineering, and Mathematics: Studies on Urban Classrooms* (pp. 268-294). New York: Information Science Reference.

[12] Cabo, C., & Lansiquot, R. D. (2014). Synergies between writing stories and writing programs in problem-solving courses. In *Proceedings of the 2014 IEEE Frontiers in Education Conference* (pp. 888-896). New York: IEEE.

[13] Whitley KN (1997). Visual Programming Languages and the empirical evidence for and against. *Journal of Visual Languages and Computing*, 8, 109-142.

[14] Haigh T, Priestley M, Rope C. (2014). Los Alamos Bets on ENIAC: Nuclear Monte Carlo Simulations, 1947-1948. *IEEE Annals of the History of Computing*, 36, 42-63.

[15] Scanlan, DA. (1989). Structured flowcharts outperformed pseudocode: An experimental comparison. *IEEE Software*, 6, 28-36.

[16] Vessey I, Weber R. (1984). Research on structures programming: An empiricist evaluation. *IEEE Transactions on Software Engineering*. 10, 397-407.

[17] Cunniff N, Taylor RP (1987) Graphical vs. Textual representation: An empirical study of novices' program comprehension. *Empirical Studies of Programmers: Second Workshop*, 114-131.

[18] Shneiderman B, Mayer R, McKay D, and Heller P. (1977) Experimental investigations of the utility of detailed flowcharts in programming. Communications of the ACM, 20(6):373-381.

[19] Ramsey HR, Atwood ME, and Van Doren JR. (1983) Flowcharts versus program design languages: An experimental comparison. Communications of the ACM, 26(6):445-449.

[20] Curtis B, Sheppard SB, Kruesi-Bailey E, Bailey J, and Boehm-Davis DA. (1989) Experimental evaluation of software documentation formats. *J. Systems and Software*, 9(2):167-207.

[21] Winslow, L. E. (1996). Programming pedagogy–a psychological overview. *SIGCSE Bulletin*, *28*(3), 17-22.

[22] Lopez M, Whalley J, Robbins P, Raymond Lister. (2008) *ICER '08 Proceedings of the Fourth International Workshop on Computing Education Research*, 101-112.

[23] Lister R, Fodge C, and Teague D. (2009) Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *ITiCSE '09 Proceedings of the 14th annual ACM SIGCSE conference on Innovation and Technology in Computer Science Education,* 161-165.