# Thinking in imperative or objects? A study on how novice programmer thinks when it comes to designing an application

Sim, Tze Ying
*School of Interdisciplinary Studies*
*Sunway University*
*Subang Jaya, Malaysia*
*tzeyings@sunway.edu.my*

***Abstract*—Novice programming is a challenging subject to teach and learn. However, programming is an essential skill that is required by many majors apart from Computer Science. The challenges in a novice programming subject change according to the programming language used. At the beginning of the 90s, the object-oriented programming was introduced. Detienne claimed that it is easier for programmers to program using the object-first approach as humans think naturally in objects. The IEEE and ACM joint task force on Computing Curriculum proposed two tracks of curriculum, one for imperative-first and the other for object-first implementation. However, most of the work conducted on novice programming focused on the issues of syntax errors, reducing the possibilities of syntax error through a new or adapted programming environment. This paper will present the preliminary work to investigate if students will naturally think in objects or a series of steps. Three intervention methods were implemented in three different workshops. The intervention methods are the object-first, the imperative-first and the problem-solving-first. The students are then requested to design an application. Through the design, the research will identify if the students use the object-first or the imperative-first design. Assuming that the object-first intervention group will design primarily in objects, and the imperative-first intervention group in a series of steps, the problem-solving intervention will be the "neutral" group. The object-first design is reflected through the attributes and methods of a particular object. The imperative-first design is identified if the solution contains a series of steps. The findings show that most of the students designed the application using a series of steps reflecting the imperative-first design. This finding should be included when considering if imperative-first or object-first should be the way forward for a novice programming subject.**

***Index Terms*—engineering education, computer science education, object oriented programming, logic programming**

## 1. Introduction

Novice programming subject refers to the first programming subject for a student. This subject is commonly known as CS1 in many institutions. The average failure rate in the class is 30% to 50% [1]. There are different challenges related to a novice programming subject [2], [3], [4], [5]. The student would need to firstly, understand and analyse the problem. Next, they would need to identify the possible solution and design the solution. Finally, implement the solution in a new language and tool. Various work had been conducted on the challenges of novice programming, for example:

- identifying the programming language that is suitable for novice programming and simplification of the language [6] [7], analysis on the error messages [8], [9], [10], [11]
- better development environment, through the use of block programming or integrated development environment [12], [13], [14], [15], [16], [17] and
- increasing the motivation for students to be involved in the subject [18], [19], [20], [21].

For the past 20 years, with the introduction of object-oriented programming (OOP), more institution is using OOP based languages as the language for the novice programming subject. Currently, there are two main camps regarding the curriculum of novice programming, the imperative-first and the object-first.

Detienne [22] mentioned that object-first should be the way forward as it is natural to think in objects. However, very few of the research looks into how a student thinks. There had been work to look into the mental model of students when they use on using specific tools, improving the design process, and concept maps [23], [24], [25], [26], [27]. There had also been some discussion if OOP languages should be used as an introductory language in a novice programming subject [18]. The objective of this research is to study how students think naturally, especially when comparing imperative-first vs object-first thinking. Understanding how students think may influence us on how to conduct a novice programming class. This research work is a part of an on-going project. The literature review will introduce a brief history of the programming languages and the work conducted in the area of the novice programming environment. Next, the methodology for this research will be presented, followed by findings and discussion. Finally, a summary and future work will be discussed.

## 2. Literature Review

A novice programmer is required to identify the problem, propose a solution, design the solution and implement the solution. The programming language will influence the

way the solution is implemented. The programming language will also affect the challenges faced by the students. The first subsection of the literature review will discuss the challenges concerning programming languages. The second subsection reviews the research work to support novice programming. The third subsection presents the work related to the mental model and skills.

## 2.1. History of Programming Languages

The challenges in novice programming due to the choice of languages had been discussed by researchers [28], [29], [30]. Each generation of programming language was designed for a specific purpose and had it's set of challenges [31].

The earliest programming language is the machine language, with bits of 1s and 0s. It is challenging to use the representation of 1s and 0s to instruct the computer to perform a specific action [32]. The next generation of language is assembly language. Assembly language uses specific syntax to represent low-level language instructions. It consists of short instruction like ADD, MOV, and SUB. Assembly language is more natural to understand as compared to the machine language. Instead of having to write the instruction in 1s and 0s, simple instructions can be used. This improves the comprehension of the code. Both machine language and assembly language are hardware dependent.

At the beginning of the 1950s, procedural languages became more popular. Procedural languages have "English like" syntax and is easier to be deciphered by a human. Among the earliest procedural languages are FORTRAN, BASIC, COBOL and C. With the feature of more English like syntax and not hardware dependent, it is more flexible for the novice programming class to be taught. The procedural languages are executed step-by-step and can be grouped into functions or procedures [33]. Procedural languages are also known as imperative language.

At the beginning of the 1990s, a new type of high-level language emerged. It is known as the OOP language. The OOP language intends to address two attributes not covered by the procedural language. They are information hiding and implementation of objects. The four concepts that encompass object-oriented programming are abstraction, polymorphism, inheritance, and encapsulation. The recommendation of "Computing Curriculum by the ACM and IEEE Joint Task Force" includes both imperative-first and object-first syllabus [34]. Detienne, an object first advocates, claim that teaching novice programming using the object-first concept is more straight forward as the world consists of objects [22].

## 2.2. Research on Novice Programming

Even though the high-level language used today are more straightforward as compared to the low-level languages, novice programmers still face problem concerning the syntax. Research on error messages faced by novice programmer concludes that most of the mistakes are due to syntax errors [10]. The other two categories of error are semantic error and type error [**?**]. Students have issues with the syntax error at the beginning of the semester, and the errors declined as the semester progress. Tools were developed to assist students when they encounter the error messages, for example, by getting help from archive solution or contacting an instructor [9]. Other ways to utilise the error messages include teaching students the debugging process [35], providing better description and solutions to the error messages [29], and generation of questions based on the error messages [8], [9], [29], [36].

Apart from working with error messages, other fellow researchers looked at eliminating the syntax error through block programming [37]. Block programming allows a novice programmer to drag and drop the required command to construct the programming code. No typing is needed, therefore eliminating the syntax errors. Examples of popular block programming are "Scratch" by Massachusetts Institute of Technology [38], [39], "Alice" by Carnegie Mellon University, and "Snap!" by University of California, Berkeley. The elimination of syntax errors allows students to concentrate on the logic of the program and students can complete the tasks sooner [16]. The tools mentioned generally do not support object-oriented programming, except for the creation of objects [40].

## 2.3. Computational Thinking

According to Andy and Gomez [41], the most challenging part of solving a problem is the problem abstraction and decomposition. Computational thinking encompasses the skills required to solve a problem in a novice programming class. Computational thinking is embedded in different field of studies [42], [43], [44]. The concepts include logical thinking, pattern recognition, abstraction, decomposition, and algorithms [45]. Decomposition is to breakdown a bigger problem to smaller parts, abstraction focuses on the general ideas, pattern recognition is identifying the similarities and differences in an issue, and algorithms is the steps and orders on how a problem can be solved. Each concept is related to each other. The ability to identify patterns, and understanding logic, can help one to propose a more efficient algorithm to solve the problem. The ability to decompose a problem will also help the student to see the bigger picture and the details of the problem. Research has shown that students who demonstrate computational skills tend to do better in a novice programming class [46].

# 3. Methodology

The objective of this project is to identify if students will "naturally" think in object or a series of steps. Three one hour session of workshops were conducted. Based on the literature review that students who demonstrate computational skills tend to do better in a novice programming class, the workshops started by introducing two computational thinking concepts logical thinking and pattern recognition. Students are then given 5 minutes to answer two questions from each category. The correct answers will be discussed after all students complete the quiz.

Next, each group was provided with activity concerning specific concepts related to novice programming. They are problem-solving, imperative-first programming, and object-first programming. The students were allocated

TABLE 1. WORKSHOP SCHEDULE

| Activities | Duration (minutes) |
|---|---|
| Opening remarks | 5 |
| Introduction to computational thinking | 5 |
| Pre-test questions | 5 |
| Problem solving/Imperative first/Object first | 15 |
| Designing an application | 15 |
| Closing remarks | 5 |
| Total duration | 50 |

15 minutes to complete the activities. The activities are described as follows:

- The problem-solving group were given two simple problems to solve. The first is to sum all the integers between 1 and 100. The second activity is provided the current year and the year of birth, derive the age of a person. They are encouraged to submit as many solutions as possible.
- The imperative-first group need to complete the ”Wonder Woman Challenge” (https://www.madewithcode.com/projects/wonderwoman). The focus here is a series of steps. There are only three puzzles. The first is a single series of actions. The second puzzle requires students to use a loop with a series of steps within and after the loop. The third puzzle put together a series of steps within a loop and introduced the condition statement.
- The object-first group need to complete the ”App Lab Introduction” activity (https://code.org/educate/applab). The students learnt the two basic concepts of an object, namely properties and methods. They were required to do simple programming to configure the properties, and actions (methods) of objects like image, text or shapes.

Following that, all three groups of students are then given the same problem to solve. They were provided paper and pen to document their solution. The students were not specifically taught how to solve the problem. They were given the statement:

In order to assist children, learn Mathematics better, you are required to help create a system that will generate Mathematics question and answers. If the questions are answered correctly, a message Good Job will appear. The questions will be displayed one by one. A final score will be displayed at the end of the ten questions.

Students were only given the instruction Work in pairs and write or sketch your solution. It need not take a specific format. You can also imagine that you are developing an application for the children. The students were only provided 15 minutes to write or sketch their solution. The solutions will be collected at the end of 15 minutes.

Finally, the last 5 minutes of the workshop was used to conclude the workshop. The summary of the activity is shown in Table 1. The duration of each workshop is 55 minutes, and the students who took part in this workshop are high school students in Year 10 and Year 11.

### 3.1. Pre-test Questions

There were four pre-test questions asked. Two questions were on logic and the other two on pattern recognition. The pre-test questions were taken from Khan Academy LSAT Preparation Test. The two selected questions are as shown in Figure 1. The two pattern recognition questions were taken from the website ”iqtestexperts.com”. They are displayed in Figure 2.

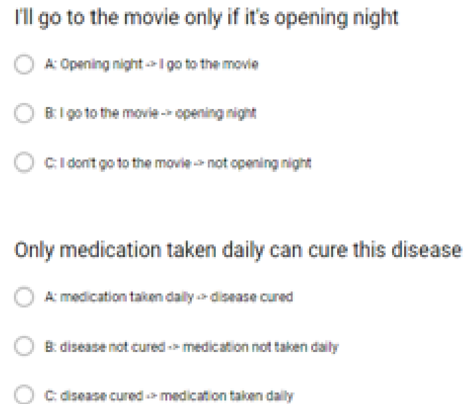


Figure 1. Logic Test Questions

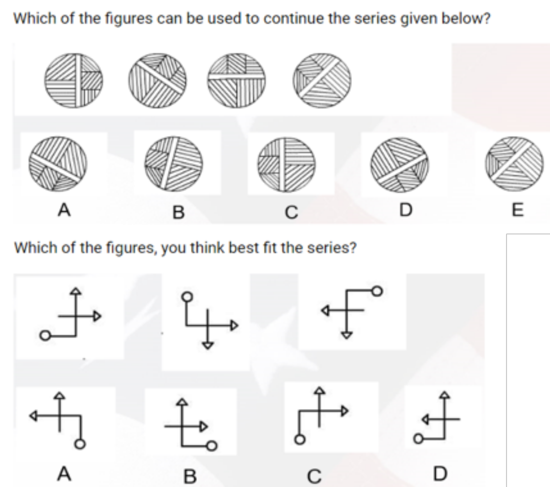


Figure 2. Pattern Recognition Questions

### 3.2. Analysis on Application Designed

Each group of students were given 15 minutes to sketch or write their design on a piece of paper. They were requested to work in pairs. However, some students choose to work individually. The frequency of each method calculates each preference. The researcher and another independent party then evaluate each piece of the paper. The design is categorized as imperative first, object first or a combination of imperative and object. The characteristic of the different categories are as below:

- Imperative first - design consist of step by step elaboration in text or diagram

- Object first - design reflects property and specific methods related to the object
- Imperative + Object - design includes an object with property or methods and step by step of the process

The most common identification of a possible object-based design is when the students design the application interface with properties and methods. Object first design elaborates on the property and actions of a specific object. Multiple screens can be used, but the property and action are clearly defined. However, if the design is a simple application interface without object and properties indication, but the screens are used to describe the different phase of the application, it will be considered as imperative first. This is because the idea behind the design is still the individual steps and the sequence on how this application should function. For the design that includes both aspects, it is considered as design with object first and imperative attributes. The following designs are considered as imperative-first design.

- Figure 3 is a sample of text describing each step of the process,
- Figure 4 describe the steps of the application using a flow chart,
- Figure 5 describes the steps using a simple graphical user interface. However, the graphical interface in figure 5 did not indicate any property or behaviour. Therefore, the interfaces are not considered to be influenced by objects. Rather, they are a series of steps expressed in graphical interfaces.

The following designs are considered a combination of object and imperative design.

- Figure 6 and figure 7 both use graphical user interface with flows indicating the steps of the application. The graphical user interface showed the action that should take place when a button is clicked. Figure 8 is text-based description. However, within the text, behaviour and property of objects can be identified.

There were no designs that reflect the only object-first design.

Step 1 display question
Step 2 check answer
Step 3 if answer correct display message " Good Job "
Step 4 display following question
Step 5 after all 10 question , check total score
Step 6 display final score

Figure 3. Sample of imperative-first design using text

## 4. Findings and Discussion

The pre-test questions serve as a baseline to the ability of students in each of the group. It is crucial to demonstrate that the students in all the categories have a similar capability in logic and pattern recognition. Generally,

Figure 4. Sample of imperative-first design using flowchart

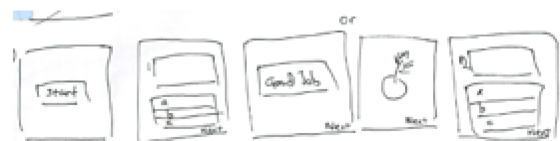


Figure 5. Sample of imperative-first design using graphical interface

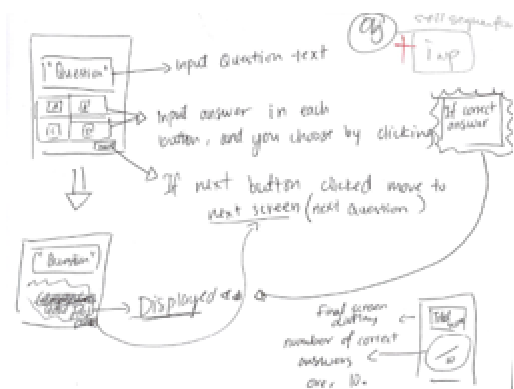


Figure 6. Detailed interfaces reflecting object and imperative design

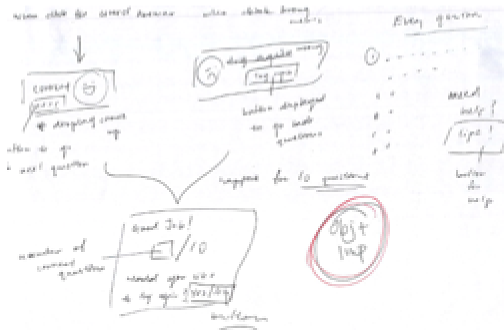

Figure 7. Basic interfaces reflecting object and imperative design

Display 10 questions for each student (multiple choice)

- Once the question appears, 4 choice of answers appear
- If the child picks the correct answer, the answer would appear green with a tick at the side
- If it is the wrong answer, the answer would appear red with an X at the side, displaying the correct answer in green
- Once all ten question is up, the screen would display which question you got wrong, and show how many you got correct
- The screen would also show "Good job", "Excellent" if they did well.
- A timer system is optional as well

Figure 8. Text description indicating behaviour of an object and the actions

most of the students scored at least three out of the four questions right (see table 2). The null hypothesis assumes that the means between the different groups are equal. Using the one-way Anova mean comparison test, the value for significance difference is 0.171. Stating the significant difference at sigma value of 0.05, the Anova test with the value of sigma 0.171 shows that there is no significant difference between the means of the different categories. Therefore, it can be assumed that the students in the different group have a similar level of capability in terms of logic and pattern recognition (see table 3).

It is possible that the students who were exposed to the imperative-first activities will prefer the imperative design, and the students who were exposed to the object-first activities will prefer the object-first design. Therefore, the problem-solving group of students will serve as the "neutral" group. From analyzing the design application of all the three groups of students, it is observed that majority of the students demonstrated a series of steps, which implied the imperative-first idea, in their design (see table 4). More than 60% each category of students demonstrated the imperative-first thoughts in the design. The imperative-first group has the highest percentage at 69%, followed by the problem-solving group at 68%, and the object-first group at 60%.

TABLE 2. PERCENTAGE OF CORRECT ANSWERS FOR PRE-TEST

| Category | 0 Correct | 1 Correct | 2 Correct | 3 Correct |
|---|---|---|---|---|
| Object First | 1 (5%) | 0 (0%) | 3 (15%) | 16 (80%) |
| Imperative First | 0 (0%) | 0 (0%) | 2 (13%) | 14 (88%) |
| Problem Solving | 0 (0%) | 3 (11%) | 8 (29%) | 17 (61%) |

TABLE 3. ONE WAY ANOVA TEST FOR PRE-TEST RESULT

| Sum | Sum of Squares | df | Mean Square | F | Sig. |
|---|---|---|---|---|---|
| Between Groups | 1.488 | 2 | 0.744 | 1.818 | 0.171 |
| Within Groups | 24.950 | 61 | 0.409 | | |
| Total | 26.437 | 63 | | | |

TABLE 4. PERCENTAGE OF CATEGORY FOR APPLICATION DESIGNED

| Category | Imperative Design | Object + Imperative Design | No Answer |
|---|---|---|---|
| Object First | 12 (60%) | 8 (40%) | 0 (0%) |
| Imperative First | 11 (69%) | 3 (13%) | 2 (13%) |
| Problem Solving | 19 (68%) | 2 (7%) | 7 (25%) |

The highest percentage of the group that demonstrated the object and imperative design, at 40%, is the group who did the object first exercise through the "App application" activities. This may due to having exposed to the way "buttons and screens" works during the "App application" exercise. Students apply what they have learnt into the design of the application. The students who did imperative-first, and problem-solving first, have 13% and 7% implementing object and imperative design, respectively. It is also interesting to observe that a majority of the student who went through the "problem solving" exercise, implemented the designs in steps, only 2 out of 28 students or 7% of the students designed in objects.

Using the one-way Anova analysis with design implemented as the factor, two mean tests were conducted. The first test, let null hypothesis be "there is no significant difference in frequencies between the imperative first, object first or no implementation". The sigma is at 0.003 (see table 5). Therefore, the null hypothesis cannot be accepted. There is a significant difference between the frequencies for imperative first, object first and no implementation.

The second test is to remove the empty design. Therefore, for the second test, let the null hypothesis be "there is no significant difference in frequencies between the imperative first and the object first design". The sigma value of the one way Anova mean comparison is at 0.22 (see table 6). Therefore, the null hypothesis cannot be accepted. There is a significant difference between the frequencies for imperative first and object first design. This finding is contrary to Detienne's claim that it is more natural to think in object [22].

TABLE 5. ONE WAY ANOVA TEST FOR IMPERATIVE, OBJECT OR NO DESIGN

| Sum | Sum of Squares | df | Mean Square | F | Sig. |
|---|---|---|---|---|---|
| Between Groups | 8.380 | 2 | 4.190 | 6.618 | 0.003 |
| Within Groups | 38.620 | 61 | 0.633 | | |
| Total | 47.000 | 63 | | | |

TABLE 6. ONE WAY ANOVA TEST FOR IMPERATIVE AND OBJECT DESIGN

| Sum | Sum of Squares | df | Mean Square | F | Sig. |
|---|---|---|---|---|---|
| Between Groups | 3.918 | 1 | 3.918 | 5.602 | 0.022 |
| Within Groups | 37.064 | 53 | 0.699 | | |
| Total | 40.982 | 54 | | | |

### 4.1. Limitation to Studies

Even though the classification was done by two independent parties, it is possible that bias could have occurred. It will be helpful for the classification to be done by more parties to ensure consistency. Therefore, the researcher provided the link to the files in the section 5 for potential reviewers.

## 5. Summary and Future work

This research reflected on the preliminary finding that it is NOT natural to think in object as proposed by Detienne [22]. Instead, when designing, it is more natural to think in steps. Majority of the students, regardless of the intervention method, implemented the imperative-first design. As fellow educators, we may want to reconsider if object-first should be the way forward for a novice programming subject. One of the challenges in the object-first implementation is novice programmers have difficulties in determining the functions and scopes of an object [47]. Since it is more natural for students to think in steps, the focus of novice programming should address the problem solving instead of adding new concepts in regards to objects.

It is also interesting to note that students who are exposed to the problem-solving-first method, which is the neutral method, are more keen to implement the imperative first design. If this a sign that we "naturally" think in steps and orders, then the novice programming subject be taught using a problem-solving method. Many novice programming class will introduce the concept before providing a student to solve the problem. There may be a difference if we reverse the process, with students having to propose a solution before the lecturers teach them the structure that corresponds to their idea. This may bring a better impact to the learning.

Moving forward, the following is planned:

- Further experimental workshops will be conducted with other groups of participants to confirm the finding of this research.
- The impact of problem solving first vs structured concepts first will be investigated.

For fellow researchers who are interested in using this set of data, the raw files are available at http://bit.ly/Object-Imperative.


## Acknowledgment

I want to thank my colleague Dr Lau, Sian Lun, for being my sparring partner to discuss the various ideas presented in this paper; and Ms Vikaneswari Shanmugan, for advising on the statistical analysis. I would also like to thank Sunway University for supporting this work through the internal grant funding.


## References


[1] C. Watson and F. W. Li, "Failure rates in introductory programming revisited," in *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE '14*, 2014, pp. 39–44.

[2] E. Lahtinen and T. Ahoniemi, "Kick-Start Activation to Novice Programming - A Visualization-Based Approach," *Electronic Notes in Theoretical Computer Science*, vol. 224, no. C, pp. 125–132, 2009.

[3] M. Piteira and C. Costa, "Computer programming and novice programmers," in *Proceedings of the Workshop on Information Systems and Design of Communication*, 2012, pp. 51–53.

[4] D. Mccall and M. Kölling, "Meaningful Categorisation of Novice Programmer Errors," in *IEEE Frontiers in Education Conference (FIE) Proceedings*, 2014, pp. 1–5.

[5] A. Luxton-Reilly, "Learning to Program is Easy," *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '16*, pp. 284–289, 2016.

[6] A. Robins, J. Rountree, N. Rountree, A. Robins, J. Rountree, and N. Rountree, "Learning and Teaching Programming : A Review and Discussion," vol. 3408, no. April, pp. 37–41, 2003.

[7] K. M. Ala-Mutka, "A Survey of Automated Assessment Approaches for Programming Assignments," Tech. Rep. 2, 2005.

[8] M. Barr, S. Holden, D. Phillips, and T. Greening, "An exploration of novice programming errors in an object-oriented environment," Tech. Rep. 4, 1999.

[9] M. Amaratunga and S. Rajapakse, "An Interactive Programming Assistance Tool ( iPAT ) for Instructors and Novice Programmers," in *The 8th International Conference on Computer Science & Education (ICCSE 2013)*, no. Iccse, Colombo, Sri Lanka, 2013, pp. 680–684.

[10] A. Altadmri and N. C. C. Brown, "37 Million Compilations," *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, pp. 522–527, 2015.

[11] B. A. Becker, G. Glanville, R. Iwashima, C. McDonnell, K. Goslin, and C. Mooney, "Effective compiler error message enhancement for novice programming students," *Computer Science Education*, vol. 26, no. 2-3, pp. 148–175, 2016.

[12] P. DePasquale, J. A. N. Lee, and M. A. Pérez-Quiñoness, "Evaluation of subsetting programming language elements in a novice's programming environment," in *Proceedings of the 35th SIGCSE technical symposium on Computer science education - SIGCSE '04*, 2004, p. 260.

[13] Y. Hosanee and S. Panchoo, "An Enhanced Software Tool to Aid Novices in Learning Object Oriented Programming (OOP)," in *Computing, Communication and Security (ICCCS), 2015 International Conference on*, Pamplemousses, 2015.

[14] M. Hu, M. Winikoff, and S. Cranefield, "Teaching Novice Programming Using Goals and Plans in a Visual Notation," p. 4352, 2012.

[15] M. Kölling, N. C. C. Brown, and A. Altadmri, "Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming," in *Proceedings of the Workshop in Primary and Secondary Computing Education*, 2015, pp. 29–38.

[16] T. W. Price and T. Barnes, "Comparing Textual and Block Interfaces in a Novice Programming Environment," *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15*, pp. 91–99, 2015.

[17] K. Roy, W. C. Rousse, and D. B. Demeritt, "Comparing the mobile novice programming environments: App Inventor for Android vs. GameSalad," in *Proceedings - Frontiers in Education Conference, FIE*, 2012.

[18] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson, "A survey of literature on the teaching of introductory programming," in *ITiCSE-WGR '07: Working group reports on ITiCSE on Innovation and technology in computer science education*, 2007, pp. 204–223.

[19] T. Howles, "A study of attrition and the use of student learning communities in the computer science introductory programming sequence," *Computer Science Education*, vol. 19, no. 1, pp. 1–13, 3 2009.

[20] A. Robins, "Learning edge momentum: a new account of outcomes in CS1," *Computer Science Education*, vol. 20, no. 1, pp. 37–71, 2010.

[21] T. Y. Sim, "Exploration on the impact of online supported methods for novice programmers," in *2015 IEEE Conference on e-Learning, e-Management and e-Services, IC3e 2015*, 2016, pp. 158–162.

[22] F. Détienne, "Assessing the cognitive consequences of the object-oriented approach: a survey of empirical research on object-oriented design by individuals and teams," *Interacting with Computers*, vol. 9, pp. 47–72, 1997.

[23] E. Dillon, M. Anderson, and M. Brown, "Comparing mental models of novice programmers when using visual and command line environments," in *Proceedings of the 50th Annual Southeast Regional Conference on - ACM-SE '12*, 2012, p. 142.

[24] L. Ma, J. Ferguson, M. Roper, and M. Wood, "Investigating and improving the models of programming concepts held by novice programmers," *Computer Science Education*, vol. 21, no. 1, pp. 57–80, 3 2011.

[25] A. Mühling, "Aggregating concept map data to investigate the knowledge of beginning CS students," *Computer Science Education*, vol. 26, no. 2-3, pp. 176–191, 2016.

[26] E. Vagianou, "Program working storage: a beginner's model," in *Proceedings of the 6th Baltic Sea conference on Computing education research Koli Calling 2006 - Baltic Sea '06*, 2006, p. 69.

[27] T. VanDeGrift, T. Caruso, N. Hill, and B. Simon, "Experience Report: Getting Novice Programmers to THINK about Improving their Software Development Process," *Proceedings of the 42nd ACM technical symposium on Computer Science Education - SIGCSE '11*, p. 493, 2011.

[28] A. Altadmri and N. C. C. Brown, "37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data," in *SIGCSE '15 Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015, pp. 522–527.

[29] B. A. Becker, G. Glanville, R. Iwashima, C. McDonnell, K. Goslin, and C. Mooney, "Effective compiler error message enhancement for novice programming students," *Computer Science Education*, vol. 26, no. 2-3, pp. 148–175, 7 2016.

[30] T. Wyeld and Z. Barbuto, "Don't Hide the Code!: Empowering Novice and Beginner Programmers Using a HTML Game Editor," *2014 18th International Conference on Information Visualisation*, pp. 125–131, 2014.

[31] J. Joque, "The Invention of the Object: Object Orientation and the Philosophical Development of Programming Languages," *Philosophy and Technology*, vol. 29, no. 4, pp. 335–356, 2016.

[32] David Hemmendinger, "Machine language," 2016.

[33] R. Decker and S. Hirshfield, "Educators' Symposium - A Case for, and an Instance of, Objects in CS1," in *Addendum to the Proceedings on Object-Oriented Programming, Systems Languages & Applications, OOPSLA*, no. October. New York: Association for Computing Machinery, 1992, pp. 3019–312.

[34] The Joint Task Force on Computing Curricula ACM IEEE-Computer Society, "Computer Science Curricula 2013 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science," ACM, IEEE Computer Society, Tech. Rep., 2013.

[35] J. H. Cross, T. D. Hendrix ', and L. A. Barowsm ', "Using the debugger as an integral part of teaching CS1," in *32d ASEE/IEEE Frontiers in Education Conference*. Boston: IEEE, 2002, pp. F1G–1– F1G–6.

[36] G. Evangelidis, V. Dagdilelis, M. Satratzemi, and V. Efopoulos, "X-compiler: Yet another integrated novice programming environment," *Proceedings - IEEE International Conference on Advanced Learning Technologies, ICALT 2001*, pp. 166–169, 2001.

[37] C. M. Lewis and C. M., "How programming environment shapes perception, learning and goals," in *Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10*. New York, New York, USA: ACM Press, 2010, p. 346.

[38] J. Maloney, K. Peppler, Y. B. Kafai, M. Resnick, and N. Rusk, "Programming by choice: urban youth learning programming with scratch," *SIGCSE '08 Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pp. 367–371, 2008.

[39] O. Meerbaum-Salant, M. Armoni, and M. M. Ben-Ari, "Learning computer science concepts with Scratch," *Computer Science Education*, vol. 23, no. 3, pp. 239–264, 9 2013.

[40] T. Y. Sim and S. L. Lau, "Online Tools to Support Novice Programming," in *IEEE Conference on e-Learning, e-Management & e-Services*, 2018.

[41] A. Gomes and A. Mendes, "A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations," in *Proceedings - Frontiers in Education Conference, FIE*, 2015.

[42] J. Walden, M. Doyle, R. Garns, and Z. Hart, "An informatics perspective on computational thinking," in *Proceedings of the 18th ACM conference on Innovation and technology in computer science education - ITiCSE '13*. New York, New York, USA: ACM Press, 2013, p. 4.

[43] B. B. Morrison, B. Dorn, and M. Friend, "Computational Thinking Bins," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19*. New York, New York, USA: ACM Press, 2019, pp. 1018–1024.

[44] L. Pollock, C. Mouza, K. R. Guidry, and K. Pusecker, "Infusing Computational Thinking Across Disciplines," pp. 435–441, 2019.

[45] Y. Dong, V. Catete, R. Jocius, N. Lytle, T. Barnes, J. Albert, D. Joshi, R. Robinson, and A. Andrews, "PRADA," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19*. New York, New York, USA: ACM Press, 2019, pp. 906–912.

[46] L. Gouws, K. Bradshaw, and P. Wentworth, "First year student performance in a test for computational thinking," p. 271, 2013.

[47] A. Ebrahimi and C. Schweikert, "Empirical study of novice programming with plans and objects," Tech. Rep. 4, 2006.