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. 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
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 its 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 straightforward 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 [2]. 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
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.

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.

Figure 3. Sample of imperative-first design using text

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

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,
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%.

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 2. Percentage of Correct Answers for Pre-test

<table>
<thead>
<tr>
<th>Category</th>
<th>0 Correct</th>
<th>1 Correct</th>
<th>2 Correct</th>
<th>3 Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object First</td>
<td>1 (5%)</td>
<td>0 (0%)</td>
<td>3 (15%)</td>
<td>16 (80%)</td>
</tr>
<tr>
<td>Imperative First</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
<td>2 (13%)</td>
<td>14 (88%)</td>
</tr>
<tr>
<td>Problem Solving</td>
<td>0 (0%)</td>
<td>3 (11%)</td>
<td>8 (29%)</td>
<td>17 (61%)</td>
</tr>
</tbody>
</table>

### Table 3. One Way Anova Test for Pre-test Result

<table>
<thead>
<tr>
<th>Sum</th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>1.488</td>
<td>2</td>
<td>0.744</td>
<td>1.818</td>
<td>0.171</td>
</tr>
<tr>
<td>Within Groups</td>
<td>24.950</td>
<td>61</td>
<td>0.409</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>26.437</td>
<td>63</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 4. Percentage of Category for Application Designed

<table>
<thead>
<tr>
<th>Category</th>
<th>Imperative Design</th>
<th>Object + Imperative Design</th>
<th>No Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object First</td>
<td>12 (60%)</td>
<td>8 (40%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Imperative First</td>
<td>11 (69%)</td>
<td>3 (13%)</td>
<td>2 (13%)</td>
</tr>
<tr>
<td>Problem Solving</td>
<td>19 (68%)</td>
<td>2 (7%)</td>
<td>7 (25%)</td>
</tr>
</tbody>
</table>
Table 5: One Way ANOVA Test for Imperative, Object or No Design

<table>
<thead>
<tr>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>8.380</td>
<td>2</td>
<td>4.190</td>
<td>6.618</td>
</tr>
<tr>
<td>Within Groups</td>
<td>38.620</td>
<td>61</td>
<td>0.633</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>47.000</td>
<td>63</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6: One Way ANOVA Test for Imperative and Object Design

<table>
<thead>
<tr>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>3.918</td>
<td>1</td>
<td>3.918</td>
<td>5.602</td>
</tr>
<tr>
<td>Within Groups</td>
<td>37.064</td>
<td>53</td>
<td>0.699</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>41.982</td>
<td>54</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
