Chang, M. et al. (Eds.) (2019). Proceedings of the 27th International Conference on Computers in Education. Taiwan: Asia-Pacific Society for Computers in Education

Investigating Functional Fixedness among Novice Student Programmers

May Marie P. TALANDRON-FELIPE* & Kent Levi A. BONIFACIO*
Central Mindanao University, Philippines
*mmptalandron@cmu.edu.ph, klab@cmu.edu.ph

Abstract: Functional Fixedness (FF) refers to a type of fixation which hinders an individual to use an object in an unfamiliar, atypical, and new ways aside from what had been previously learned about the object's functionality. In a problem-solving activity, FF may delay or impede the process of finding a solution due to the inability to utilize the available materials and options presented. In this study, the incidence of FF in the context of computer programming was investigated. The students were given programming problems that require them to utilize a specific Java pre-defined method in a different manner from its basic and usual implementation. Results showed that all students exhibited at least 1 incidence of FF although no significant relationship was found between FF and their poor performance in solving the problems. The students explained that they got stuck in the known usage and it was difficult for them to think of other ways to use it in order to solve the problems.

Keywords: Functional Fixedness, Computer Programming, Novice Programmer

1. Introduction

Functional Fixedness (FF) was first hypothesized by Duncker (1945) which refers to the tendency of being fixed to specific uses or functionality of an object based on prior knowledge, how the objects were introduced, or the level of familiarity with the object. This fixation may then hinder the usage of the object in a different, atypical, or new manner which may be essential in a problem-solving task.

In the context of programming, researchers linked FF to cognitive issues in teaching students how to code functions (Mutsuddi & Frame, 2018) and loops (Smith, Paradice, & Smith, 2000). Schwill (1994) once recommended to teach programming using an object-oriented paradigm in the introductory level because it reflects the fundamental cognitive process of human by relating functions and usage to an object. He added that it should be done in ways that students could explore how to utilize object analogies of programming concepts beyond their basic definition. In coding user-defined functions, students show fixation with the function structure so when presented with sample integer functions, there was a difficulty coding user-defined functions for string. Similarly, in teaching the usage of for-loops using a series of examples where loops were used to sum elements, the students were fixated on the specific usage (i.e. summing elements) that when asked to write a looping statement for a different purpose, they tend to get stuck. However, these studies did not present an actual programming experiment where students could exhibit FF.

This study aims to revisit the concept of functional fixedness and investigate its incidence among student programmers taking up an introductory programming course and its implication on the students' problem-solving performance on programming problems.

2. Functional Fixedness

One trait that distinguishes humans from other creatures is that humans are creative beings by nature (Ward, Smith, & Vaid, 1997). This trait contributes to the ability of humans to find solutions to problems in addition to general intelligence. However, some psychological phenomena could impede the extent of creativity and one of them was described by cognitive psychologists as fixation. One reason for fixation is functional fixedness which refers to the inability to use an object in a different manner other than what had been previously known about it (Duncker, 1945). Several studies (Flavell,

Cooper, & Loiselle, 1958; Glucksberg & Danks, 1968; Jensen, 1960) further showed that functional fixedness occurs when the function of a key object is explicitly demonstrated, taught, or explained prior to the presentation of a problem that requires the utilization of the object in a way that was not taught.

One of the several experiments conducted to support this phenomenon was called the "box" problem (Duncker, 1945; Frank & Ramscar, 2003). The subject was presented with different materials including candles, matches, thumbtacks and boxes of varying sizes. The task was to fix the candles on the wall and light them. The solution is to tack the boxes to the wall using the thumbtacks and these boxes would serve as the bases for the candles. Then the candles are lit and fixed on the boxes with some wax. For the first group, the materials were presented in a way that the boxes were used as container for the candles, thumbtacks, and matches. For the second group, the materials were just scattered on the table. More subjects from the second group solved the problem than from the first group. Duncker explained that this is because the subjects from the first group perceived the boxes just as containers and it was difficult for them to lose the function associated with it.

In the context of computer programming, Perkins and Martin (1986) have previously explored this among novice programmers and argued that FF is a matter of negative knowledge transfer wherein knowing "how" sometimes impairs rather than support one's performance in the application of a solution in an unfamiliar context. Another related phenomenon to FF called the Einstellung Effect (EE) which refers to an individual's bias towards a familiar, working solution (Luchins, 1942; Luchins & Luchins, 1961) has been investigated among programming students (Obispo, Castro, & Rodrigo, 2018). EE is also a type of fixation and surprisingly, the study showed that, in a single programming activity, EE had a positive effect on the performance of the student programmers.

However, FF is different from EE in that FF deals with separate functionality of the objects and not a collective solution such in the case of EE. And since there have been no recent empirical studies to investigate the implications of FF on the performance of student programmers, this warrants further investigation.

3. Experiment

The participants of this investigative study were thirty (30) freshmen students taking up Bachelor of Science in Information Technology (BSIT) from a state university in southern Philippines. They were all considered as novice programmers based on their programming grades and also because they were only on their first year in the BSIT program. They were presented with the 3 Java pre-defined methods: Java String indexOf() and startsWith() as well Java Integer.parseInt() which they have to use to solve 3 programming problems. These methods had already been taught to them in prior lectures in their programming classes.

The use of both Integer.parseInt() and string.startsWith() are straightforward but the participants must realize the string.indexOf() could be used in a different way other than just finding an index of a character or string. They were instructed that they will use these methods along with the basic program control structures in order to solve the following programming problems:

Problem1: A string s1 contains various integers separated by spaces. s1 = "50 1 822 3 6 49 5 61 74 87" The task is to save the individual integers delimited by space into an integer array called myArr.

Problem 2: Ask the user to input a full name. Then, ask the user to input a prefix composed of 3 characters. The program should then check if the last name from the first input starts with the prefix.

Problem 3: Ask the user to input a string. Check if the string includes the substring "love".

There could be several approaches to solve each of the three problems using the given methods. After the experiment, the students were then interviewed on how they felt about the programming activity where they discussed their solutions and their thoughts about using the given methods.

4. Results

The incidence of FF was based on whether the student was able to utilize indexOf() to solve the problem. One FF point was given if the student failed to use of indexOf() in order to solve the programming problem. Program codes submitted by the participants for each problem was analyzed in order to measure their performance and to determine the incidence of FF. Aside from looking at the

program logic, the codes were also checked for syntax and runtime errors to account for other possible reasons in cases where the student was unable to solve the problem. A point was given only if the program was able to execute and perform the task required in the problem and the indexOf() method was used as part of the solution. Only 6 students were able to earn points from successfully solving the programming problems but all students exhibited at least 1 incident of functional fixedness.

When the relationship between the FF incidence and student performance in terms of solving the problems was investigated, no significant relationship was found $r(30)=-0.19$, $p=0.51$. However, after submitting their program codes, the students were interviewed about the programming activity and the concept of functional fixedness was explained to them. The students reasoned that they got stuck in the known examples demonstrating the basic and usual usage of the indexOf() method and they cannot think of other ways to use it that will help solve the problem.

In conclusion, FF does exist among novice computer programmers because students are not yet adept in the creative ways on how these pre-defined methods could be used in various programming purposes aside from those that they have previously seen from examples. Although no significant relationship was found with performance in the experiment, the fact that a total of only 6 students out of 30 were able to solve at least 1 problem because others cannot think of how to use a previously known Java pre-defined method to solve the problem means that demonstrating and encouraging creativity is essential as a pedagogical technique in classes for novice student programmers. The authors would like to recommend further experiments and analysis that may involve intermediate programmers, more pre-define methods, and other programming components.

Acknowledgements

The authors would like to thank the students from the Institute of Computer Applications who participated in the experiment.

References

Duncker, K. (1945). On problem-solving. *Psychological Monographs*, *58*(5).

Flavell, J. H., Cooper, A., & Loiselle, R. H. (1958). Effect of the number of pre-utilization functions on functional fixedness in problem solving. *Psychological Reports*, *4*(3), 343–350.

Frank, M. C., & Ramscar, M. (2003). How do presentation and context influence representation for functional fixedness tasks? *Proceedings of the Annual Meeting of the Cognitive Science Society*, *25*.

Glucksberg, S., & Danks, J. H. (1968). *Effects of discriminative labels and of nonsense labels upon availability of novel function*. *7*, 72–76.

Jensen, J. (1960). On functional fixedness: Some critical remarks. *Scandinavian Journal of Psychology*, *1*(1), 157–162.

Luchins, A. S. (1942). Mechanization in problem solving: The effect of Einstellung. *Psychological Monographs*, *54*(6), i.

Luchins, A. S., & Luchins, E. H. (1961). Einstellung effect in social learning. *The Journal of Social Psychology*, *55*(1), 59–66.

Mutsuddi, A., & Frame, D. (2018). Can Cognitive Science make it Easier for Novice Students to Learn to Code Functions? *Proceedings of the 19th Annual SIG Conference on Information Technology Education*, 149. International World Wide Web Conferences Steering Committee.

Obispo, J. R. C., Castro, F. E. V. C. G., & Rodrigo, M. M. T. (2018). Incidence of Einstellung Effect among Programming Students and its Relationship with Achievement. *Proceedings of the 1st Information Computing Education*. Presented at the Information Computing Education, Philippines.

Perkins, D. N., & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. *At Empirical Studies of Programmers, 1st Workshop, Washington, DC*, 213–229.

Schwill, A. (1994). Cognitive aspects of object-oriented programming. *IFIP WG 3.1 Working Conference "Integrating Information Technology into Education*.

Smith, D. K., Paradice, D. B., & Smith, S. M. (2000). Prepare your mind for creativity. *Communications of the ACM*, *43*(7), 110–116.

Ward, T. B., Smith, S. M., & Vaid, J. (1997). Conceptual Structures and Processes in Creative Thought. In *Creative Thought: An Investigation of Conceptual Structures and Processes* (pp. 1–27). Washington, D.C.: American Psychological Association.