Investigating the Under-Usage of Code Decomposition and Reuse Among High School Students: The Case of Functions

Ahmad Omar[1(✉)], Irit Hadar[1(✉)], and Uri Leron[2]

[1] Information Systems Department, University of Haifa, Haifa, Israel
ahmadomar3@gmail.com, hadari@is.haifa.ac.il
[2] Faculty of Technology and Science Education, Technion, Haifa, Israel
urileron@gmail.com

Abstract. Functions can provide substantial benefits for programmers. They offer ways that can be used to simplify a given programming task through decomposition, reusability and abstraction. As observed by the first author, a graduate student and high school computer science (CS) teacher, students do not spontaneously use functions when they are asked to solve a certain task; instead they provide one procedural solution, even in situations where functions can clearly be helpful. This research aims to investigate how and when students use functions, as well as the reasons underlying their decisions whether to use them. This paper presents our ongoing research including some results from a pilot study. For data analysis we use the dual-process theory of human cognition and three related concepts: comfort zone, principle of least effort and cognitive laziness. We discuss how these can be useful in order to better understand the problem at hand.

Keywords: Programming · Functions · Reusability · Decomposition · Abstraction dual-process theory · Comfort zone · Principle of least effort · Cognitive laziness

1 Introduction

Decomposition in programming refers to the process of breaking down a higher-level problem into sub-problems, allowing programmers to focus independently on each sub-problem [22]. Functions, procedures and methods are form of abstractions, serving at the same time as decomposition mechanism. They allow dividing a given problem into several simpler tasks, then combining them together for the full solution [19].

Programmers have been observed to under-use different forms of abstractions, thus not fulfilling their potential benefits [11, 12]. In this research, we approach the population of high-school students in the early phases of learning programming, in order to investigate what it is that leads programmers, from the very start of their familiarity and experience with code development, to under-use functions, one of the most basic forms of abstraction. More specifically, the objective of this ongoing research is to investigate how and when students use functions, as well as the reasons underlying their decisions whether to use them. Preliminary results obtained via a pilot study, demonstrate that students do not use functions unless they are explicitly instructed to do so, despite their evident familiarity with functions and their proper use. In order to investigate the reasons underlying this behavior, we borrowed a theory and related concepts from cognitive psychology research: the dual-process theory, cognitive laziness, the principle of least effort and, and comfort zone. In this paper we demonstrate how they can useful for making sense of the data obtained.

2 Related Work

Not using functions is not a mistake per-se, but can be inefficient in many cases. It is one of the basic mechanisms for code modularization, decomposition and reusability: “Modularization allows one to decompose a system into functional units, to impose hierarchical ordering on functional usage, to implement data abstractions, and to develop independently useful subsystems” [13]. High modularity allows for separation of concerns, namely dealing separately with each module’s details while ignoring other modules’ detail, and later with all features of all modules and the relationships between the modules, in order to combine them into a coherent system.

Code modularity facilitates code reuse, namely the ability to use parts of a computer program written previously, aiming to increase productivity and quality in large-scale software development projects [7].

Code modularity mechanisms, such as functions, are forms of abstraction. Abstraction in computer science(CS) involves throwing away detail while keeping the essential structure, and is a skill that is hard to master [1, 14]. Kramer [18] suggests that it is abstraction that differentiates good from weaker students. Functions, in particular, are widely accepted to be a difficult concept to learn [20].

Previous works studying programming skills of high school students have mainly focused on investigating, exploring and reviewing programming mistakes [5, 23]. For example, Brown [5] lists 18 of the most common student mistakes, which vary from simple mistakes, as in the case of incorrect semicolon at the end of a method header, to more complicated ones, such as not controlling the program flow properly. But programming mistakes are not only done by students; even experienced software developers make mistakes, such as confusing inheritance direction or failing to identify objects or classes [11, 12], demonstrating the resilience or such difficulties.

Other directions of research focused on the processes involved in teaching or in learning to program, and on how to improve these processes for better results. Such proposed improvements include, for example, having teachers present real-life examples [3]; letting the students write pseudo-code solutions in order to focus more on the structure and not be held back by syntax [8]; allocating more time in class for exercises, with the teacher as guider [2]; and, letting students learn from their mistakes [10]. No such improvement technique was examined in the specific context of functions.

3 Theoretical Background

The dual-process theory deals with the question of why people make mistakes that could have been avoided given their own knowledge [16]. The theory suggests that two separate cognitive systems operate in our mind: intuitive and analytical thinking. The first system (S1) deals with immediate, automatic thinking processes, based on heuristics. The second system (S2) is responsible for analytical processing. S1 results in fast and automatic solutions, while S2 in slower, consciously analyzed ones [16].

Cognitive laziness refers to the situation in which people may be content with a "good enough" solution, despite their awareness that a better solution can be achieved if they invest more effort [9]. Although there may be a better solution, people tend to choose the solution that is enough for their immediate goals. Intuitive judgment and common sense are examples of mechanism used due to cognitive laziness [21].

The principle of least effort explains that people (or even animals or smart machines) will naturally choose the path of least effort, with the least resistance, having the desire to reach things quickly and easily [25]. Accordingly, a person facing a certain situation, tends to choose the solution with the least effort from all possible solutions rising in the horizon (as perceived by that specific person) [6].

Comfort zone is a situation in which a person behaves in an anxiety neutral condition, and acts limitedly, in order to sustain a steady, risk-free state of tasks' performance [24]. The term comfort zone refers to a psychological state in which a person feels comfortable, at ease and in control [24], and suggests that individuals may choose the less stressful and challenging solution, in order to be in the more comfortable and less adventurous state, from their point of view [4].

4 Pilot Study – Method and Settings

The objectives of this research are to empirically explore our initial observation that students do not use functions when they deal with programming tasks, unless they are explicitly asked to do so, and to understand the reasons underlying this behavior.

The population of pilot study included 10 students of the 12th grade, who majored in CS in the school in which the first author teaches. These students had already learned and exercised the use of functions and were trained in programming tasks.

During the study, the students had free access to their books and notebooks. Each student was handed three worksheets which included programming tasks. The worksheets were given one at a time; the next sheet given only after the participants completed the previous one, in order to prevent them from checking questions in the next worksheet thus possibly affecting the way they solve the questions in the current one.

The first worksheet included three tasks, with a shared functionality to all, but with no further information. It instructed to add a number to all items in a given array, with the tasks differing only in the value of the added number. The second worksheet included three tasks of the same level with a shared functionality and an instruction, at the bottom, asking participants to pay attention to what is common in the tasks. The third worksheet also included three tasks of the same level, with an instruction to write a function for the common functionality and to use it to solve the three tasks.

Data collection was based mainly on: (1) Written solutions collected from the participants after each task; and (2) group interviews [17] about their solutions. We used a semi-structured approach with no rigorous set of questions, so to allow diversion [15].

In analyzing the worksheets' answers, we looked for general solutions, rather than fully correct ones; we did not consider syntax or any other compilation errors but rather considered the intention behind the students' answers. We classified the answers to those using functions and those that do not. During the data analysis, we looked for explanations for the obtained results. Specifically, in analyzing the qualitative data from the transcribed group interview, we mapped students' quotes to a corresponding cognitive concept based on an initial set of guidelines we developed:

Table 1. Preliminary guidelines for mapping explanations and cognitive concepts

Explanation type (self-report)	Mapped to
A student did not use functions because s/he did not feel comfortable enough with it, despite being aware to this possibility	Comfort Zone
A student considered different alternatives including functions, and settled, consciously for a "good enough" solution[a]	Cognitive laziness
A student considered different alternatives, including functions, and chose, consciously, the possibility that was perceived to save effort (cognitive effort, time, writing, etc.)	Principle of least effort
A student wrote the first things that came up to mind, without considering other possibilities	General – dual-process theory

[a]The "good enough" solution is not necessarily the one that requires the least effort; in fact, using functions requires the least effort overall (writing less code). When *cognitive laziness* takes over, it leads to a "good enough" solution considering the least effort in the *short* term (merely for solving the first task).

5 Results

In worksheet1, all students used a *for* loop, for each task, while changing the value of the added number each time. Not even a single student wrote a function for the common functionality for reusing it in the different tasks. When the students were later whether they had noticed a common functionality between the tasks, one student raised his hand. When asked why he did nothing about it, he answered: "I didn't know what to do with this common functionality; I just solved the question as required."

In worksheet2, all students used *for* loops, the same way as in worksheet1, even though they were asked here to pay attention to what is common between the tasks. When asked about their solutions, all students stated that they had noticed the comment about the commonality. When asked why they did not use a function instead of repeating the same code with a change of value, they gave the following explanations:

1. "I wrote the first thing that crossed my mind."

2. "I answered without thinking too much; it was almost automatic."
3. "I answered in the easiest way."
4. "It is more comfortable for me to solve it this way."
5. "The number of questions is small, why the effort?"
6. "I answered in the simplest way, straightforward, without complication."
7. "I answered in the way with the least possibility to make mistakes."
8. "The tasks did not require us to write a function. Had that been a requirement in the assignment, I would have no problem doing that."
9. "There were only few questions and the questions themselves were easy."

In worksheet3, the students were asked explicitly to write a specific function and to use it in solving the set of tasks given in this worksheet. All the students, with no exception, wrote the function properly and used it in solving the three questions.

When they were asked if they had faced any difficulty writing the function or using it, they all answered with a no. When we further asked them why, unlike in the previous two worksheets, in this one they had wrote and used a function for the common functionality, they explained that it was because they were asked to do so.

Trying to make sense of the observation that students do not take advantage of functions and reuse, despite their proven knowledge and capability to do so, we mapped their explanation quotes to the four cognitive psychological concepts according to the guidelines presented in Table 1. The mapping is presented in Table 2.

Table 2. Theory and related quotes

Cognitive theory/concept	Related quotes
Comfort zone	4, 7
Cognitive laziness	6, 8
Principle of least effort	3, 5, 9
Dual processing theory	1, 2

This demonstration of mapping students' explanations to the cognitive concepts presents some promise toward an understanding of the different sources leading to the avoidance of using functions. The next step of the research will involve a higher-volume data collection to allow for a more quantitative investigation and generalization of the results. In addition, since students' explanations do not necessarily accurately reflect their actual thinking processes, we plan to use the think-aloud protocol as a complementary method in the research in order to mitigate this threat.

6 Discussion and Future Work

The pilot study included a small number of participants, all belonging to the same school. The full study will include 12th grade students from four high schools, with the participation of about 30 students from each school. The schools will be of different levels, different teaching languages, and different populations: two private schools and two public ones, with Arabic or Hebrew as teaching languages. These settings are designed so to improve the external validity of the findings.

As a result of this research we hope to expend our understanding on the underlying cognitive processes that lead to the under-usage of functions. Such understanding may lay the foundations for developing means for promoting functions' use, by overcome the identified barriers for using functions. We will also provide a set of guidelines for mapping given behavior to cognitive states possibly triggering this behavior. These guidelines could be used in the context of function usage in programming specifically, or generally when investigating any under-used tool or information that cannot be explained by the investigated individuals' lack of knowledge.

References

1. Aharoni, D., Leron, U.: Abstraction is hard in computer-science too. In: Pehkonen, E. (ed.) Proceedings of the 21st Conference of the International Group for the Psychology of Mathematics Education. University of Helsinki, Lahti, Finland (1997)
2. Black, T.R.: Helping novice programming students succeed. J. Comput. Sci. Coll. **22**(2), 109–114 (2006)
3. Börstler, J., Hall, M.S., Nordström, M., Paterson, J.H., Sanders, K., Schulte, C., Thomas, L.: An evaluation of object oriented example programs in introductory programming textbooks. In: ACM SIGCSE Bulletin, vol. 41(4), pp. 126–143 (2010)
4. Brown, M.: Comfort zone: model or metaphor? J. Outdoor Environ. Educ. **12**(1), 3 (2008)
5. Brown, N. C., Altadmri, A.: Investigating novice programming mistakes: educator beliefs vs. student data. In: Proceedings of the 10th Annual Conference on International Computing Education Research, pp. 43–50. ACM (2014)
6. Collan, M.: Lazy user behavior, MPRA Paper No. 4330 (2007). http://mpra.ub.uni-muenchen.de/4330/
7. da Silva, M.F., Werner, C.L.: Packaging reusable components using patterns and hypermedia. In: Proceedings of 4th International Conference Software Reuse, pp. 146–155. IEEE (1996)
8. Fidge, C., Teague, D.: Losing their marbles: syntax-free programming for assessing problem-solving skills. In: Proceedings of the 11th Australasian Conference on Computing Education, vol. 95, pp. 75–82. Australian Computer Society, Inc. (2009)
9. Fiske, S.T.: Thinking is for doing: portraits of social cognition from daguerreotype to laserphoto. J. Pers. Soc. Psychol. **63**(6), 877 (1992)
10. Ginat, D.: The greedy trap and learning from mistakes. In: ACM SIGCSE Bulletin, vol. 35(1), pp. 11–15 (2003). ACM
11. Hadar, I.: When intuition and logic clash: the case of the object-oriented paradigm. Sci. Comput. Program. **78**(9), 1407–1426 (2013)
12. Hadar, I., Leron, U.: How intuitive is object-oriented design? Commun. ACM **51**(5), 41–46 (2008)
13. Hashim, K., Key, E.: A software maintainability attributes model. Malays. J. Comput. Sci. **9**(2), 92–97 (1996)
14. Hazzan, O., Kramer, J.: Assessing abstraction skills. Commun. ACM **59**(12), 43–45 (2016)
15. Hove, S.E., Anda, B.: Experiences from conducting semi-structured interviews in empirical software engineering research. In: 11th IEEE International Symposium on Software Metrics (2005)
16. Kahneman, D.: Maps of bounded rationality: a perspective on intuitive judgment and choice. Nobel Prize Lecture **8**, 351–401 (2002)

17. Kontio, J., Lehtola, L., Bragge, J.: Using the focus group method in software engineering: obtaining practitioner and user experiences. In: Proceedings of the International Symposium Empirical Software Engineering ISESE 2004, pp. 271–280. IEEE (2004)
18. Kramer, J.: Is abstraction the key to computing? Commun. ACM **50**(4), 36–42 (2007)
19. Meyer, B.: Object-Oriented Software Construction. Prentice Hall, New York (1988)
20. Paz, T., Leron, U.: The slippery road from actions on objects to functions and variables. J. Res. Math. Educ. **40**, 18–39 (2009)
21. Pearl, J.: Heuristics: intelligent search strategies for computer problem solving (1984)
22. Rosson, M.B., Alpert, S.R.: The cognitive consequences of object-oriented design. Hum. Comput. Interac. **5**(4), 345–379 (1990)
23. Sirkiä, T., Sorva, J.: Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. In: Proceedings of the 12th Koli Calling International Conference on Computing Education Research, pp. 19–28. ACM (2012)
24. White, A.: From comfort zone to performance management. White & MacLean Publishing (2009)
25. Zipf, G.K.: Human behavior and the principle of least effort: An introduction to human ecology. Ravenio Books (2016)