The Subjective Cost of Writing Reusable Code: The Case of Functions

Itamar Lachman[1], Irit Hadar[1(✉)], and Uri Hertz[2]

[1] Department of Information Systems, University of Haifa, Haifa, Israel
Itamar.lachman@gmail.com, hadari@is.haifa.ac.il
[2] Department of Cognitive Sciences, University of Haifa, Haifa, Israel
uhertz@is.haifa.ac.il

Abstract. Functions provide substantial benefits for software development, simplifying programming through decomposition, reusability and abstraction. In a previous study, our group identified a tendency of high-school students to not use functions, even in programming tasks where functions can be a good solution strategy. The current research extends this observation to university students and aims to provide an explanation for the factors underlying this tendency. We focus on the subjective cost of the cognitive effort required for writing functions. Our experiment examined how information systems students solved a set of programming tasks, which varied by the number of repetitive questions. The results showed that most of the students avoided using functions altogether. We further found that in the subgroup of students who used functions at least once, the likelihood of using functions was positively associated with (a) the number of repetitive questions in each task, and (b) the task order, i.e., the progress of the experiment. These results indicate that the subjective cost of writing functions is taken into account when making a decision on how to solve a task at hand and is compared with the cost of repetitive work without using function, and that the former cost is updated with experience.

Keywords: Programming · Functions · Code-reuse · Abstraction · Cognition · Dual-process theory · Subjective cost

1 Introduction

Software development requires the ability to design and write code using abstract constructs, such as functions and classes. This requires to think abstractly, a skill which is not easily mastered [5, 9]. For example, Hadar [4] showed that while developers possess the knowledge about the concepts and principles of object-oriented design (OOD), they demonstrate design errors, some of which stem from the need to think in abstract terms. The explanation offered for these errors is heuristics thinking [7], reflected in developers' intuition about objects in the real world, which in some cases conflicts with the formal principles of object-oriented design (OOD).

Omar at el. [12] found that when asked to write code for simple tasks, in which the use of functions is a good strategy, high-school students did not spontaneously use functions unless they were explicitly asked to do so, even when they had proven

H. A. Proper and J. Stirna (Eds.): CAiSE 2019 Workshops, LNBIP 349, pp. 29–39, 2019.
https://doi.org/10.1007/978-3-030-20948-3_3

knowledge and ability to use them successfully. A qualitative analysis of the reasons the students provided for their decision why they did not use functions, indicated three main reasons: (1) The principle of least effort [8, 18] – choosing the easiest solution available of the perceived options. (2) The principle of satisficing [14, 15] – choosing the first solution that comes to mind which seems good enough. (3) Comfort zone [1] – choosing familiar habits. All three identified reasons are closely related to habitual and intuitive thinking, as explained by the dual-process theory [7].

In this study, we extend the previous research [12] about how cognitive factors affect the decision whether to use functions when writing code. First, we extend the population of high-school students to university students. Second, we seek a more in-depth explanation as to how the process of deciding whether to use functions is affected by the effort required in the different tasks. More specifically, our objective is to understand which factors lead developers to migrate from the strategy of not using functions to that of using them. Finally, we examine whether learning occurs over the course of repetitive decisions as to the use functions, thus affecting future decisions.

Our approach combines an additional cognitive perspective – the value-based subjective cost of effort [16] – with the aforementioned explanation of the dual-process theory. According to this approach, when faced with a coding task, the developer contemplates different solutions and evaluates their subjective cost in terms of cognitive effort. For example, not using functions may incur little effort in non-repetitive tasks but will be very costly when the task is highly repetitive. This estimation is based on the developer's previous experience or assumptions and may change after additional experience is gained. The developer compares the different subjective values of the alternative coding solutions, e.g. using functions and not using functions, to decide on her course of action.

Guided by this approach, we address the following research questions:

RQ (1): How will the number of questions (repetitions) in each coding task influence the use of functions in solving these tasks?
RQ (2): Can we observe an order effect (experience) on the use of functions?

The remainder of this paper is organized as follows: Sect. 2 provides some theoretical background on the cognitive theories used in this research. Section 3 describes the research method and Sect. 4 presents its results. Section 5 discusses the results and threats to validity and Sect. 6 concludes.

2 Theoretical Background

2.1 The Dual-Process Theory

The phenomenon investigated in this paper, namely the selection of the a satisfactory solution to a problem even when inferior to an alternative solution, is related to Herbert Simon's study on bounded rationality and concept of satisficing [14]. Bounded rationality is the idea that decision makers' rationality is limited by the tractability of the decision problem, the cognitive limitations of their mind, and the time available to

make the decision. As a result, they will seek a satisfactory, good enough, solution rather than an optimal one.

This idea was also examined in Kahneman's Nobel Prize lecture (2002), "Maps of bounded rationality: A perspective on intuitive judgment and choice," relying on the dual-process theory. According to this theory, two distinct systems operate in our minds: an intuition-based system (S1) and a reasoning-based system (S2). The first (S1) is characterized as fast, parallel, with automatic, effortless and associative slow-learning heuristics, whereas the other (S2) is characterized as a slow, serial, controlled, effortful, rule-governed and flexible.

The relation of this model to the idea of least cognitive effort is that the purpose of humans' S1 is to reduce the cognitive effort required by S2. When we need to do things that can be done automatically, mostly habitual tasks we have already acquired some knowledge of how to perform, we prefer to do them by using the S1-based intuitive thinking [7]. This preference of S1 over S2 results in some of the principles explained previously, for example, the availability heuristic, i.e. selecting the first answer that comes to mind, which also adds to its presumed correctness ("it feels right") [7]. Based on this theory, we predict that functions would be less used (unless the participant has previously used them habitually), because using functions requires abstract thinking involving S2, which in turn requires a more effortful thought process than that involved in S1 thinking of the dual-process model [7].

2.2 Subjective Cost of Cognitive Effort

The idea of subjectivity of an option's cost or value (i.e., subjective value) draws from studies in behavioral economics. Subjective values have been studied in the context of delayed reward [6], for example, where decision makers preferred smaller amounts of money delivered immediately over higher sums delivered in some future time. In the context of cognitive effort, participants were shown to prefer tasks which were less cognitive demanding but were associated with low rewards, over more cognitively demanding tasks which incurred high rewards [16]. In another study participants implicitly learned to choose between two decks of cards of task instructions, one of which included more cognitively demanding tasks than the other [8]. In these studies, the experimenters parametrically manipulated the reward associated with high cognitive load (or future delay) to track the participants' indifference point, e.g. how much reward should be associated with a demanding task to make the participant choose it over an easy task on 50% of the trials. This indifference value represented the participants' individual subjective cost associated with cognitive effort.

These results demonstrate the idea that people try to avoid cognitive effort, such that high cognitive effort is associated with higher subjective cost. This behavior may bear a resemblance to our investigation on the decision of using functions. If we harness the idea that the subjective cost will increase as the need for cognitive effort arises, then we can assume that because functions require abstract thinking, their subjective cost will be higher than using an alternative option that does not require this type of thinking (for example, writing loops). It is therefore reasonable to assume that when people choose between the two options of writing code, one using functions and

the other using loops, they would prefer the option to which they allocate a lower subjective cost of effort.

Moreover, we assume that tasks that involve more work will be considered as having higher subjective cost of effort. At the same time, tasks that require abstract thinking will be considered as having higher subjective cost as well. However, while the subjective cost of using functions does not change when the number of repetitive questions change, the subjective cost of using loops is to be considered based upon the amount of work required, which is derived from the number of repetitions in the tasks. Accordingly, our aim is to identify the point in which the subjective cost of using loops becomes higher than that of using functions.

3 Method

3.1 Experiment Design

The experiment was designed to test the following hypotheses:

For **RQ (1):** How will the number of questions in each coding task influence the use of functions in solving these tasks?

H (0): There will be no correlation between the number of questions in each task and the frequency of use of functions.
H (1): The frequency of use of functions will be positively correlated with the number of questions in each task.

For **RQ (2):** Can we observe an order effect (experience) on the use of functions?

H (0): There will be no correlation between the task order and the frequency of the use of functions.
H (1): There will be a positive correlation between the task order and the frequency of the use of functions.

The experiment consisted of an introduction explaining the experiment to the participants, a precondition part aimed at ensuring that the participants have the knowledge needed for developing code using functions, and the main part of the experiment including a series of 15 coding tasks. Participants were asked to use a coding language of their choice, or pseudo code. Participants were able to use copy and paste within a task but copying text between tasks was restricted by the experiment, in order to isolate the effort invested in each task.

The precondition part included a single task in which the participants were asked to write a function that adds a value to an array of elements, in a coding language of their choice. While all our participants completed at least one academic programming course (C), this part allowed us to identify and exclude participants who nevertheless still struggle with the proper use of functions.

The functions part included 15 simple arithmetic tasks on arrays which included: adding, multiplying or subtracting each of the elements of multiple arrays, as follows:

- **Add:** consists commands to perform the operation of adding each time a different number to each of the elements of a new array.
- **Subtract:** consists commands to perform the operation of subtracting each time a different number from each of the elements of a different array.
- **Multiply:** consist commands to perform the operation of multiply each time a different number with each of the elements of a different array.

Each of the add, subtract and multiply operations were used in tasks which included 1, 3, 5, 8 or 15 questions (number of arrays), resulting in 15 tasks all together (3 operations × 5 number of questions). The order of the tasks was randomized for each participant. Table 1 demonstrates the type of actions and reoccurrence patterns of the tasks.

Table 1. Examples of tasks

Type	Examples of a 3 questions (repetitions) tasks
Add	1. Please add 24 to each element in array A 2. Please add 11 to each element in array B 3. Please add 35 to each element in array C
Subtract	1. Please subtract 24 out of each element in array A 2. Please subtract 56 out of each element in array B 3. Please subtract 7 out of each element in array C
Multiply	1. Please multiply each element in array A by 12 2. Please multiply each element in array B by 44 3. Please multiply each element in array C by 75

3.2 Participants

The experiment was executed at the University of Haifa, during the spring of semester of 2018, in an object-oriented programming course. The participants were 42 students in their 1st year of study in the department of Information Systems. The participants included 24 males and 18 females, of ages 19–29 with the average age of 21. The study was approved by the institutional ethics committee, and participants gave their informed consent to participate.

3.3 Procedure

The experiment took place in a computer lab classroom. Each of the participants was provided with a computer and a link to the experiment website, a general explanation about the experiment and the reward for completing it (extra credit in the course).

Upon entering the website, each participant filled a consent form and registered into the experiment, filling non-identifiable data only (e.g., age, course, gender and coding experience). Then the participants were presented with a short introduction including an explanation about the series of tasks of code writing they are about to receive. This

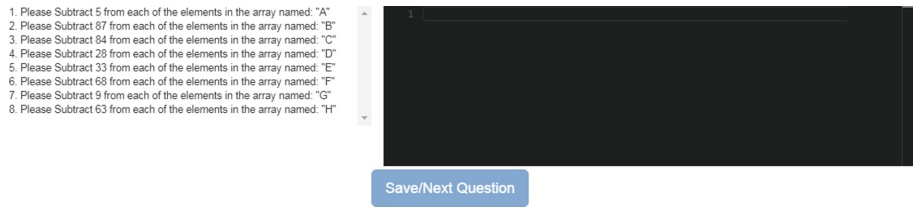

Fig. 1. Example of a coding task

was followed by the precondition task, and then by the series of tasks, as detailed above (see example in Fig. 1).

3.4 Data Analysis

The evaluation of function use in the participants' answers (code) in both the precondition and the main experimental tasks was performed according to the following criteria: for the code to be considered a proper function, it had to be surrounded by a block name and at least two parameters, and to be called from another block of code. Two external evaluators evaluated the use of functions in the answers of the participants. An additional criterion employed was that for a solution to be evaluated as a function, both evaluators' answers needed be positive.

First, a screening of the precondition task was performed, in order to exclude participants who failed in this task. Following the evaluation of this task, one participant was excluded, leaving 41 participants included in the dataset. Next, the evaluators evaluated the full set of the experiment tasks for the 41 participants. A Pearson correlation ($r = 0.85$, $n = 40$, $p = 0.02$) between the evaluators and Kappa Cohen Test ($k = 0.85$) for nominal scales were calculated, indicating high agreement level.

4 Results

Analyzing the answers elicited in the experiment, we first identified the following three main patterns:

1. "Always use functions" (3/41 participants) – in this behavior pattern, the participants chose to write a function in the first question that followed the precondition task and continued using functions in each of the tasks throughout the experiment. See an example solution with the use of function in Fig. 2.
2. "Never use functions" (22/41 participants) – in this behavior pattern, the participants did not use functions in any of the questions. See example solution without the use of function in Fig. 3.
3. "Sometimes use functions" (16/41 participants) – in this behavior pattern, the participants used functions from time to time during the experiment.

This pattern of results reveals that most of our participants (see Fig. 4) never used functions, even when the task included 15 repetitive questions. This finding supports

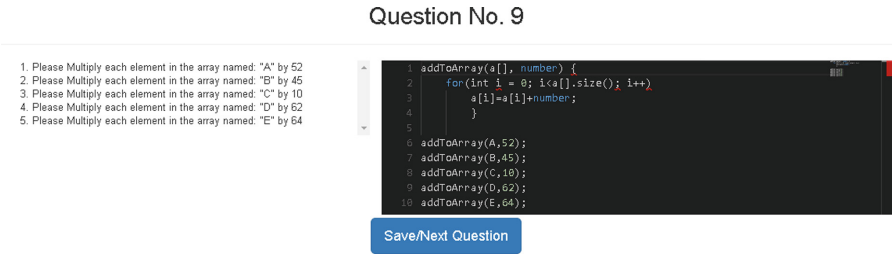

Fig. 2. Example of a solution with the use of function

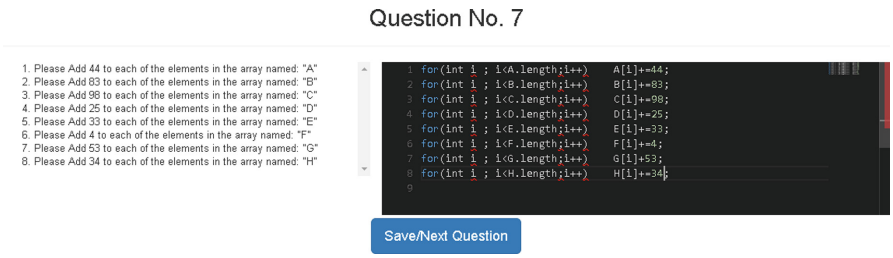

Fig. 3. Example of a solution without the use of function

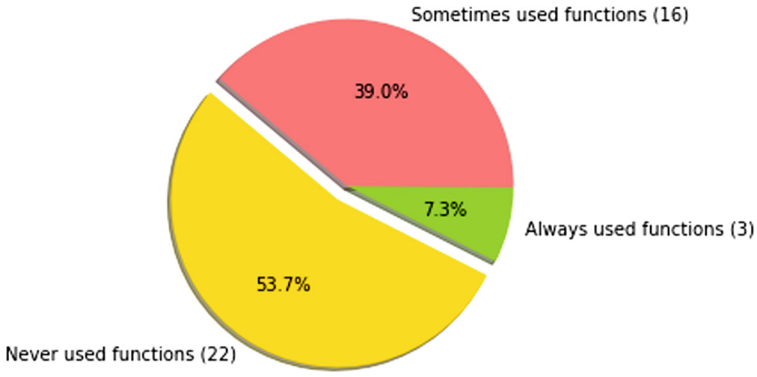

Fig. 4. Patterns of function usage

the findings of Omar et al. [12], and further generalizes them for increased repetitiveness and to the population of university students. We discuss these findings in Sect. 5.

We next focused on the subgroup of participants who used functions from time to time. We used a mixed-effect logistic regression [11] to evaluate the effects of the number of questions in a task and task order on their decisions to use functions.

We used the trial-by-trial decision to use functions using this decision (a binary parameter: 1 = function used, 0 = no function) as a dependent variable, and the number of questions (1, 3, 5, 8, 15) and task number (order in the experiment, 1–15) as fixed-effects independent variables, and the participants' ordinal number as a random effect independent variable. The model's AIC score was 373, and a null model's AIC score (a model with only the intercept random effect) was 453. The likelihoods of these two models were used to calculate the McFadden's pseudo-R squared for the fixed effects [10]: $R^2 = 0.19$.

We found that the number of questions had a positive effect on the decision to use functions (Estimate ± Standard Error (SE) = 0.22 ± 0.06, z = 3.29, $p < 0.001$). This effect confirms our hypothesis for RQ (1), H (1), as participants' frequency of using functions increased with the number of questions (Fig. 5 includes means and SEs across participants).

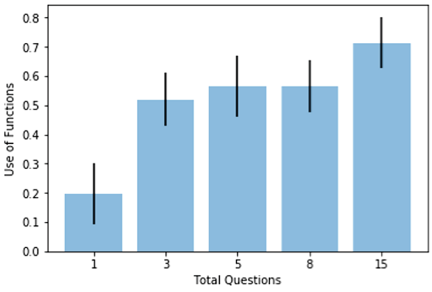

Fig. 5. Effect of number of questions in a task on the use of functions

Our analysis also revealed a positive effect of task order on the probability of using function (Estimate ± SE = 0.21 ± 0.05, z = 3.89, $p < 0.001$). This result confirms our hypothesis for RQ (2): H(1), as participants were more likely to use functions later in the experiment (Fig. 6 includes means and SEs across participants).

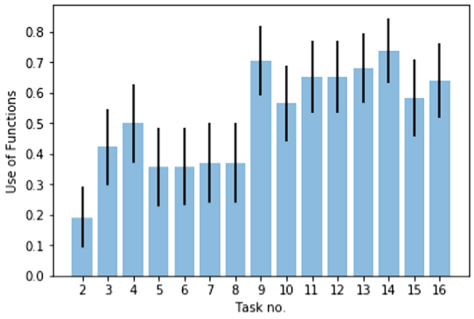

Fig. 6. Effect of order on using functions

5 Discussion

In this experiment, we aimed to evaluate the subjective cost of function use of participants performing simple coding tasks, in which functions are a good strategy. In order to evaluate the subjective cost of writing functions, we controlled the number of repetitive questions in each task.

Our main hypothesis was that by increasing the number of questions, the cost of avoiding functions will at some point meet and exceed the subjective cost of using functions, and participants will migrate from not using to using functions. This means that the likelihood of using functions will increase with the number of questions (repetitions) in a task. We based this hypothesis on the idea that in each of the tasks, unless the participants have already chosen to use functions, their default habitual solution will be to use loops. However, as the number of questions increases so will the number of loops required to complete the task, and thereby the subjective cost of this task. As a result, in some of the tasks, we assumed that the subjective cost of coding the loops will be high enough so that the participants will consider the alternative solution of using functions.

We also hypothesised that each decision that the participants will make in each of the tasks will contribute to their experience about the benefits of using functions instead of loops, and thus that there will be a temporal learning effect in which as the number of previous tasks increases, so does the likelihood of the participants to use functions.

Based on the results of the experiment presented above, we identified several interesting behavior patterns. First, we found that most participants did not use functions at all, even in tasks which included 15 repeated questions, and even after writing a similar function successfully in the precondition task. This pattern amplifies the effect found by Omar et al. [12], where high-school students avoided using functions in a one-shot design. This pattern of thinking resembles the heuristic-based decision of S1 in the dual-process theory [6], which supports habitual and automatic behavior, without deliberation and reasoning. These participants might be following a habit of using loops, which they may perceive as a good enough solution.

Another subset of our participants did use functions from time to time. These participants' decision to use functions was found to be dependent on the number of repetitive questions in a task and the task order in the experiment, in line with our hypotheses and the concept of subjective cost [3, 8, 13, 16, 17]. This pattern of results reflects a subjective cost pattern, in which the cost of function use is compared with the cost of not using functions on a trial-by-trial basis. When the task is highly repetitive, the cost of not using function exceeds the cost of using functions, and therefore the probability of using functions increases. The cost of using function decreases with experience, e.g. after using functions in previous tasks, making the use of functions a more compelling option. The effort of designing the function reduces after this solution has already been used. The mixed pattern, which we view as subjective learning, resembles more the thinking process that may occur in S2, in which there is an analytical comparison of the two options and the subjective cost of using functions is estimated and compared to the alternative of writing loops.

Lastly, we assume, but can currently not confirm, that the reason that some of the participants decided to use only functions and no loops, could be the result of an earlier epiphany effect caused by the precondition task of the experiment [2]. According to this assumption, when the participants were asked to perform the precondition task in which they were explicitly instructed to use a function, some of them may have grasped the benefit of using functions and used it from this point onwards.

5.1 Threats to Validity

Construct validity: Concerns the relationships between theory and observation. In this study, while some of the participants displayed a decision behavior of subjective cost, most of them (about 54%) chose to avoid using functions altogether. We can therefore not be confident with the subjective cost of effort being a sole explanation for our research question regarding the factors that affect the decision to use functions.

External validity: Concerns the generalization of the results. The main threats in this area stem from the fact that our multi-tasks experimental design permitted using simple tasks such as mathematical operations on arrays that do not match the real-life demands and complexity of the environment of the software development industry.

6 Conclusion

In this work we aimed to investigate the cognitive factors affecting writing reusable code, specifically in the case of functions. We found that a subset of the participants reflected a subjective cost pattern, namely a decision based on comparing the cognitive effort to be invested in each alternative, in line with the concept of subjective cost [3, 8, 13, 16, 17]. These results demonstrate the manifestation of the concept of subjective cost, originated from economic decision making [3, 17], in the domain of software engineering, focused here on programing decisions. Most of the participants chose to not use functions at all. We propose that this pattern is not based on the subjective cost of the tasks but rather on habits and heuristical thinking [7]. Taken together, these results shed light on the different cognitive factors and mechanisms underlying programmers' decision making.

Future work may investigate the factors that lead to the different behavior patterns observed in this study. An understanding of these factors may inform future efforts to design strategies that would lead students and developers to more frequently chose to program code in a reusable manner.

References

1. Brown, M.: Comfort Zone: model or metaphor? Aust. J. Outdoor Educ. **12**, 3–12 (2008)
2. Chen, W.J., Krajbich, I.: Computational modeling of epiphany learning. Proc. Natl. Acad. Sci. **114**, 4637–4642 (2017). https://doi.org/10.1073/pnas.1618161114
3. Green, L., Myerson, J.: A discounting framework for choice with delayed and probabilistic rewards. Psychol. Bull. **130**, 769–792 (2004). https://doi.org/10.1037/0033-2909.130.5.769

4. Hadar, I.: When intuition and logic clash: the case of the object-oriented paradigm. Sci. Comput. Program. **78**, 1407–1426 (2013). https://doi.org/10.1016/j.scico.2012.10.006
5. Hashim, K., Key, E.: A software maintainability attributes model. Malays. J. Comput. Sci. **9**, 92–97 (1996)
6. Kable, J.W., Glimcher, P.W.: The neural correlates of subjective value during intertemporal choice. Nat. Neurosci. **10**, 1625–1633 (2007). https://doi.org/10.1038/nn2007
7. Kahneman, D.: Maps of bounded rationality: a perspective on intuitive judgment and choice. Sveriges Riksbank Prize Econ. Sci. Mem. Alfred Nobel, 449–489 (2002). https://doi.org/10.1037/0003-066x.58.9.697
8. Kool, W., McGuire, J.T., Rosen, Z.B., Botvinick, M.M.: Decision making and the avoidance of cognitive demand. J. Exp. Psychol. Gen. **139**, 665–682 (2010). https://doi.org/10.1037/a0020198
9. Kramer, J.: Is abstraction the key to computing? Commun. ACM **50**, 36–42 (2007). https://doi.org/10.1145/1232743.1232745
10. Maddala, G.S.: Limited-Dependent and Qualitative Variables in Econometrics. Cambridge University Press, Cambridge (1986)
11. Manlove, K.: Introduction to Statistical Analysis using R commander, pp. 1–23 (2014)
12. Omar, A., Hadar, I., Leron, U.: Investigating the under-usage of code decomposition and reuse among high school students: the case of functions. In: Metzger, A., Persson, A. (eds.) CAiSE 2017. LNBIP, vol. 286, pp. 92–98. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60048-2_9
13. Patzelt, E.H., Kool, W., Millner, A.J., Gershman, S.J., Note, A., Patzelt, E.H.: Building N In press at Scientific Reports 4729
14. Simon, H.A.: Administrative behaviour. Aust. J. Public Adm. (1947). https://doi.org/10.1111/j.1467-8500.1950.tb01679.x
15. Stanovich, K.E.: What Intelligence Tests Miss: The Psychology of Rational Thought. Yale University Press, New Haven (2009)
16. Westbrook, A., Braver, T.S.: Cognitive effort: a neuroeconomic approach. Cogn. Affect Behav. Neurosci. **15**, 395–415 (2015). https://doi.org/10.3758/s13415-015-0334-y
17. Westbrook, A., Kester, D., Braver, T.S.: What is the subjective cost of cognitive effort? Load, trait, and aging effects revealed by economic preference. PLoS One **8**, 1–8 (2013). https://doi.org/10.1371/journal.pone.0068210
18. Zipf, G.K.: Human Behaviour and the Principle of Least Effort: An Introduction to Human Ecology, 588 pages. Addison-Wesley Press, Cambridge (1949)