Predicting Algorithmic Complexity for Individuals

Sara Todorovikj (sara.todorovikj@hsw.tu-chemnitz.de)
Daniel Brand (daniel.brand@hsw.tu-chemnitz.de)
Marco Ragni (marco.ragni@hsw.tu-chemnitz.de)
Department Behavioural and Social Sciences, Technische Universität Chemnitz,
Straße der Nationen 62, 09111 Chemnitz

Abstract

How difficult is it to simulate an algorithm in one's mind and correctly deduce its outcome? In this paper, we present a predictive modeling task in the domain of algorithmic thinking in a railway environment. We present metrics, either based on algorithmic representation complexity (e.g. lines of code) or on the effect on cognitive resources an algorithm simulation can have (e.g. context switching). We implement the metrics within a benchmark and evaluate their predictive performance on an individual level, by assigning a complexity threshold to each individual. We compare these results to a standard statistical correlation analysis and suggest a different perspective for determining the predictive powers of complexity metrics as models.

Keywords: Algorithmic thinking; predictive modeling; problem solving; cognitive processes; deduction

Introduction

An algorithm is a set of well-defined instructions to be followed in order to solve a specific class of problems. Even though algorithms are most often associated with mathematical and computer sciences, they are present in each human's everyday life. Food recipes, furniture building instructions, getting coffee from a coffee machine... Each one of these examples has a set of rules associated with them that we follow in order to obtain the desired outcome.

Imagine you are building your new bookshelf, and the next step in the visual instructions depicts hammering all the nails in the package, totaling up to 40. What if the instructions were written? One way to express this step would be to write the instruction "Hammer a nail" 40 times, which is obviously unreasonable. Instead, there would be a condensed version represented as a *loop of operations* which would indicate that the hammering action needs to be repeated 40 times, or, while we still have nails in the package. Such loops are very often encountered in computer programming and we distinguish two sorts (Rogers, 1967): *for*-loops where the instructions are repeated *for* a certain amount of times and *while*-loops where the instructions are repeated *while* a specified condition holds.

The comprehension and formulation of algorithms has been researched by psychologists investigating computational thinking (Bucciarelli, Mackiewicz, Khemlani, & Johnson-Laird, 2022). Creating an algorithm for solving a problem requires solving representative instances of the problem class, simulating the process of solution to abduce an algorithm and simulating an algorithm to deduce its consequences in order to determine its correctness (Khemlani, Mackiewicz, Bucciarelli, & Johnson-Laird, 2013).

We narrow down the algorithm domain to a *railway environment*, following Khemlani et al. (2013). Given an ordered sequence of train wagons on a track, rearrangement algorithms of different complexities can be executed, leading to a new order of the wagons on a different track. Focusing on deducing an algorithm's output in this domain, we are interested in the difficulty of such tasks, operationalized by the correctness that humans achieve when trying to solve them. Khemlani et al. (2013) present their finding that for deduction, the difficulty does not depend on the number of moves performed while executing an algorithm, but rather on the Kolmogorov complexity of a corresponding Lisp function containing *while*-loops for rearranging trains of any length. In this paper, our interest lies in using complexity metrics to model the difficulty an individual has when deducing the correct wagon order after applying a rearranging algorithm.

Though often used as synonyms, the terms 'complexity' and 'difficulty' are in fact used to differentiate between (1) intrinsic characteristics which influence performance but are independent of the context and the people solving the problem, and (2) the direct relationship to the observed performance and subjective individual experience (Effenberger, Cechák, & Pelánek, 2019). There exists a relationship between complexity and difficulty, which, as shown in various settings, is reflected in a human's behavior and performance (Sheard et al., 2013; Campbell, 1988; Liu & Li, 2012).

The complexity of an algorithmic task can be defined using different metrics, like the previously mentioned Kolmogorov complexity (Khemlani et al., 2013), or simply the length of lines in a code describing the algorithm (Nguyen, Deeds-Rubin, Tan, & Boehm, 2007). To further explore the relation between complexity and difficulty, we introduce several metrics, where some are based on algorithmic complexities, while others take the cognitive perspective into consideration, specifically the effect on cognitive resources that the simulation of an algorithm can have.

In order to assess whether our metrics reflect the relationship between complexity and difficulty, we conducted an experiment to establish a data foundation. The experiment was based on the tasks in Khemlani et al.'s (2013) study, where participants were asked to deduce the outcome of applying different rearrangement algorithms to a set of ordered

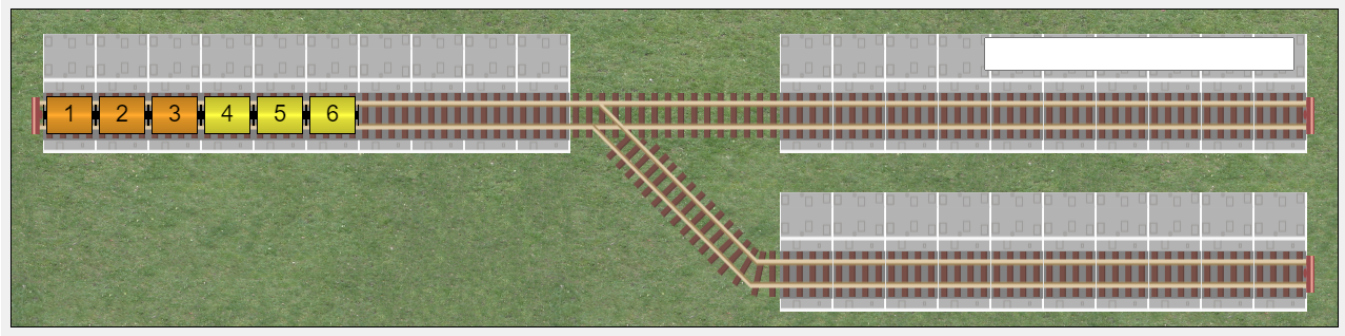

Figure 1: The *Top Left*, *Top Right* and *Bottom Right* train tracks.

wagons. Following Goodwin and Johnson-Laird (2011), we first evaluated the metrics based on the correlation between their values and the correctness achieved by participants. In a second step, we defined a modeling task where the objective was to predict the correctness of the participants' answers. By implementing our metrics as models within this task, we evaluated their predictive performance and investigated whether the results found based on correlations translate to an individual predictive level.

Experiment

The goal of the experiment was to test the ability of individuals to correctly deduce the consequence of applying an algorithm. Given a sequence of wagons in a particular order and a rearrangement algorithm, the participants' goal was to simulate the algorithm's execution in their mind and deduce the final order of the wagons, which they provided as an answer.

Our task and algorithm design was inspired by Khemlani et al. (2013). The task scenario consisted of three train tracks (*Top Left*, *Top Right* and *Bottom Right*), as shown in Figure 1. Initially, the wagons were placed in the *Top Left* track, and the goal was to have the wagons rearranged in the *Top Right* track. The *Bottom Right* track could be used by the algorithms as an intermediate track.

The four algorithms that we used from the railway domain are *Reverse*, *Palindrome*, *Parity Sort* and *Faro Shuffle*. Table 2 shows the initial states of the wagons and their final order after applying the algorithms to four and six wagons. We visualized the algorithms shown in Figure 2 using code blocks (see Figure 3). We distinguish between three types of code blocks - `While`, `Repeat` and `Move`. The `While` blocks indicate that the commands in their scope will be repeatedly executed as long as a condition holds and they are equivalent to *while*-loops. The `Repeat` blocks work similarly to `While`, except the number of repeated executions is explicitly determined with an integer, equivalent to *for*-loops. The `Move` blocks describe the wagon moving operation that should be performed. A wagon can be moved only between *Top Left* and *Bottom Right*, and *Top Left* and *Top Right*.

The experiment manipulated the type of the rearrangement algorithm and the number of wagons that the algorithm needs to be applied to (four or six). This allows for examining the difficulty of a deduction not only based on the algorithm itself, but also by taking into consideration how an algorithm's complexity changes when the number of instances it needs to be applied to differs. Finally, every participant was presented with eight tasks - four algorithms applied once to four wagons, and once to six.

Reverse

```
While track Top-Left has wagons:
    Move wagon from Top-Left to Bottom-Right
While track Bottom-Right has wagons:
    Move wagon from Bottom-Right to Top-Left
    Move wagon from Top-Left to Top-Right
```

Palindrome

```
While first wagon on Top-Left has color blue:
    Move wagon from Top-Left to Bottom-Right
While track Bottom-Right has wagons:
    Move wagon from Bottom-Right to Top-Left
    Repeat 2 times:
        Move wagon from Top-Left to Top-Right
```

Parity Sort

```
While track Top-Left has wagons:
    Move wagon from Top-Left to Top-Right
    Move wagon from Top-Left to Bottom-Right
While track Bottom-Right has wagons:
    Move wagon from Bottom-Right to Top-Left
While track Top-Left has wagons:
    Move wagon from Top-Left to Top-Right
```

Faro Shuffle

```
While not solved:
    Move wagon from Top-Left to Top-Right
    While first wagon on Top-Left has color orange:
        Move wagon from Top-Left to Bottom-Right
    Move wagon from Top-Left to Top-Right
    While track Bottom-Right has wagons:
        Move wagon from Bottom-Right to Top-Left
```

Figure 2: Wagon rearrangement algorithms as presented in the experiment. The algorithms were visualized using code blocks, as shown in Figure 3.

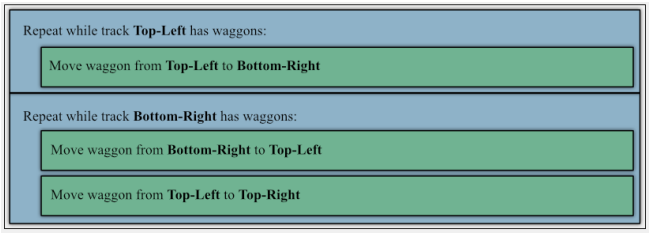

Figure 3: Example code blocks for the *Reverse* algorithm.

Participants

Thirty-six participants completed the experiment (age 18-35, 72% female). They were recruited on Prolific[1] and the experiment was performed online as a web-experiment. After completing the experiment, participants received a compensation of 10 EUR. 21 of them indicated to have 'some' programming background, 14 had 'none' and 1 participant had 'profound' background. All of them were native English speakers.

Procedure

Participants were first given an introductory task, which explained the visualization of the train tracks and the code blocks that describe the algorithm for rearranging the wagons. They were presented with the following two rules regarding the execution of the `Move` block: 1. Only one wagon will be moved at a time; 2. Only the wagon closest to the crossing on a track will be moved (it is referred to as *the first wagon of the track*). They were informed that their goal is determining the order of the wagons on the *Top Right* track resulting from executing the algorithm described by the code blocks. They needed to write their answer in a text-field above the *Top Right* track. Participants were instructed not to use external tools (like pen and paper) to solve the task, but were encouraged to simulate the algorithm in their mind. Afterwards, the participants received their first task. They were presented with either four or six wagons on the *Top Left* track and code blocks for a rearrangement algorithm. Once they entered their answer, they could proceed to the next task.

Observed Data

The total number of tasks in the experimental data is 288 (36 participants, 8 tasks each). We eliminated 12 tasks, because while solving the task, participants left the page for more than half a minute in total, leaving us with 276 valid data points (*Reverse*: 71, *Palindrome*: 69, *Parity Sort*: 69, *Faro Shuffle*: 67). For 66 of them (23.9%) participants provided correct answers (*Reverse*: 24, *Palindrome*: 11, *Parity Sort*: 18, *Faro Shuffle*: 13). The exact number of correct answers for each one of the eight tasks is provided in Table 1.

Modeling Difficulty

Based on the data obtained from our experiment, we introduce a modeling task for algorithmic thinking within the railway environment. The objective of the modeling task is to determine the difficulty, operationalized by the number of errors participants make deducing an algorithm's outcome based on the algorithm and the initial arrangement.

[1] `https://www.prolific.co/`

Table 1: Total number of valid data points and correct answers for each task.

Algorithm	#Wagons	Total Valid	Correct
Reverse	4	36	10 (27.8%)
	6	35	14 (40.0%)
Palindrome	4	35	4 (11.4%)
	6	34	7 (20.6%)
Parity Sort	4	35	9 (25.7%)
	6	34	9 (26.5&)
Faro Shuffle	4	34	11 (32.4%)
	6	33	2 (6.0%)

Complexity Metrics

In the following we introduce seven metrics which can be roughly divided into two conceptual groups. The first group of metrics are based on the complexity of the algorithm's structure only, which is common for assessing the complexity of program code. They are suited to represent an individual's ability to understand the algorithm and the underlying concept of what the algorithm is supposed to do and estimate the complexity of the execution based on the complexity of the algorithm itself. Metrics of the second group consider the precise steps performed by the algorithm when executed on a specific output. Therefore, these metrics can better account for the cognitive load that occurs while an individual simulates the steps of the algorithm in their mind, but also requires them to execute the algorithm in order to measure the respective complexity estimate.

Depth is based on algorithmic computational complexity which increases when nesting loops. Assuming that when the innermost statements are deeper nested in loops, simulating the algorithm's execution should be more difficult for an individual, the metric describes the depth of an algorithm with respect to (nested) loops, while starting with a top-level depth value of 1. *Reverse* has a depth value of 3, obtained by adding 1 (top-level) + 1 (first level - `While` commands), + 1 (second level - instructions within `While` commands). This metric provides the same value for an algorithm, independent of the number of wagons it would be applied to.

Structure mimics the relation between the length of a code describing an algorithm and a perceived level of task difficulty (Sheard et al., 2013; Nguyen et al., 2007) by counting the number of code blocks in the algorithm. *Reverse* has a structure value of 5, as it has five blocks. Similarly to the depth metric, structure also provides the same value for an algorithm, for any number of wagons.

Moves is the number of wagon moves that the algorithm performs until completion, following Khemlani et al. (2013). When applying *Reverse* to four wagons, 12 moves are

Table 2: Initial and goal states, and complexity metric values for each one of the eight railway domain tasks.

Algorithm	Initial	Goal	Complexity Metrics						
			Depth	Structure	Moves	Commands	Contexts	Signature	Entropy
Reverse	1234	4321	3	5	12	22	8	5.625	0.822
	123456	654321			18	32	12	5.906	0.833
Palindrome	1234	1423	4	6	8	18	4	5.25	0.929
	123456	162534			12	26	6	6.125	1.022
Parity Sort	1234	1324	3	7	8	17	5	6	1.015
	123456	135246			12	24	7	7	1.074
Faro Shuffle	1234	1324	4	7	6	15	5	6	0.937
	123456	142536			12	28	9	11	1.050

performed: 4 (first `Move` instruction on four wagons) + 2 × 4 (two `Move` instructions on four wagons).

Commands takes the structure metric a step further and counts the amount of times that code blocks have been executed, thereby acknowledging possible costs for checking a loop's condition. In the four wagon scenario, *Reverse* has a commands value of 22: 8 (first `While` and its `Move` blocks are executed once for each wagon) + 12 (second `While` and two `Move` blocks executed for each wagon) + 2 (execution of two `While` blocks when their condition does not hold).

Contexts represents the people's limitation to attending to only one context in their working memory and the cognitive load increase when context switching is necessary (Garavan, 1998). This metric defines a context as operating on a pair of tracks, i.e. moving wagons from one track to another. When switching between different `Move` instructions, the relevant pair of tracks changes which leads to a context switch. The metric counts the number of context switch occurrences during an algorithm execution, where a higher number indicates higher cognitive load and therefore a task is deemed more difficult. In the case of *Reverse* with four wagons, 8 context switches happen: 1 (operating on *Top Left* and *Bottom Right* in the first `While` block and switching to *Bottom Right* and *Top Left* in the second `While` block) + 7 (constant alternating between *Bottom Right* and *Top Left* and *Top Left* and *Top Right* when executing the second `While` block).

Signature imitates the repetition effect (Bertelson, 1961), which shows that an individual needs less time to perform a repeated task. We transform the effect to a complexity metric in this domain, by assuming that once an individual has processed a command once, its repetitions within a loop should be perceived as easier. The metric assigns a cost of 1 to each executed command in an algorithm, while checking if a command is immediately repeated (within a loop), in which case the cost is halved in each repetition. The signature cost of *Reverse* on four wagons is 5.625: 1.875 (first `Move` command repeated four times: 1 + 0.5 + 0.25 + 0.125) + 3.75 (other two `Move` commands repeated four times).

Entropy is a measurement of potential knowledge and randomness in information theory (Shannon, 1948). Used as a metric to quantify uncertainty, we apply it in this scenario by taking into consideration the distribution of the wagons over all three tracks. After each move, the entropy of the wagons on the tracks is calculated, as shown in Eq. 1. A higher entropy value indicates a more chaotic distribution of the wagons, potentially increasing the difficulty for individuals to simulate the algorithm and deduce its correct outcome. The final value is the average of all calculated entropies. *Reverse*'s entropy when applied to four wagons is 0.822.

$$E = -\sum_{k}(p_k \cdot \log_2 p_k) \tag{1}$$

The complexity metrics' values for each task are presented in Table 2.

Following approaches in related work (Goodwin & Johnson-Laird, 2011; Khemlani et al., 2013; Khemlani, Goodwin, & Johnson-Laird, 2015), we measure the correlation between the complexity metrics' values and the correctness of the individuals' answers, results shown in Table 3.

Complexity Metric	ρ	p-value
Depth	-0.145	**.016**
Structure	-0.008	.149
Moves	0.078	.202
Commands	0.019	.751
Contexts	0.086	.152
Signature	-0.133	**.027**
Entropy	-0.123	**.041**

Table 3: Correlation (Pearson's ρ) between complexity metrics and answer correctness. Significant p-values are marked in bold.

Modeling Individuals

The significant correlation values (Table 3) indicate that depth, signature and entropy should be the best predictors of answer correctness. We want to determine whether this holds on the individual level - are the metrics with statistically significant correlation good predictors of task difficulty for each individual in our data set?

In our modeling approach models are evaluated based on their ability to account for an individual's capability to correctly deduce the final order of the wagons after applying a rearrangement algorithm. To perform our evaluation, we relied on the Cognitive Computation for Behavioral Reasoning Analysis (CCOBRA) framework[2], which facilitates model evaluations with a focus on modeling reasoning behavior on the individual level (Riesterer, Brand, & Ragni, 2020). Similar to Riesterer et al. (2020), we performed a coverage analysis, which allows models to fit to each individual participant in the data. This approach allows to assess a models ability to represent a participant's behavior within its parameter space. In our case, the models were created by equipping each metric with a complexity threshold that represented the maximum complexity that an individual participant could "handle", i.e., the complexity value up until which the participant is able to give the correct answer. All models then fitted their thresholds to each individual. When a model is then queried for a prediction for a given task, it determines its prediction by comparing the individual's threshold to the task complexity according to the respective metric: If the complexity is too high, it is predicted that the individual will not solve this task correctly.

Besides models for each metrics, we implemented an additional baseline model which always predicts the participant to give an incorrect answer. This serves as a reasonable lower-bound, as the average correctness for the tasks was below 50%.

Results

Each model was judged on its ability to account for an individual's difficulty threshold, on which it depends whether a correct answer is given or not. Table 4 shows the accuracy values for each complexity metric and Figure 4 shows how good the individual participants are predicted. All of them achieve an accuracy above 80% and perform better than the baseline model.

The best performance is achieved by entropy, closely followed by structure, with an accuracy value of 87%. Interestingly, entropy showed a significant correlation to the answer correctness, yet structure did not. The signature metric, with a significant correlation performed very well in our benchmark, reaching an accuracy of 86%. However, even though its mean performance is rather high, Figure 4 shows that it does not manage to fully cover as many individuals as the other metrics.

[2] https://github.com/CognitiveComputationLab/ccobra

The discrepancy between significant correlations and predictive powers is shown by the depth metric - the worst predictor out of all seven metrics, even though it has a significant correlation. A part of the problem might be the dichotomous nature of the metric, which only assigns the values 3 or 4 to the present algorithms. This restricts the expressiveness of the threshold, impeding its ability to discern between individuals. On the other hand, the structure metric is still the second best predictor without a significant correlation value, although it only distinguishes between 3 possible values for our tasks.

Complexity Metric Model	Accuracy
Entropy	87%
Structure	87%
Signature	86%
Contexts	83%
Commands	83%
Moves	83%
Depth	81%
Baseline	76%

Table 4: Benchmark evaluation results - accuracy values of the complexity metrics as predictive models for a task's difficulty, ordered from best to worst.

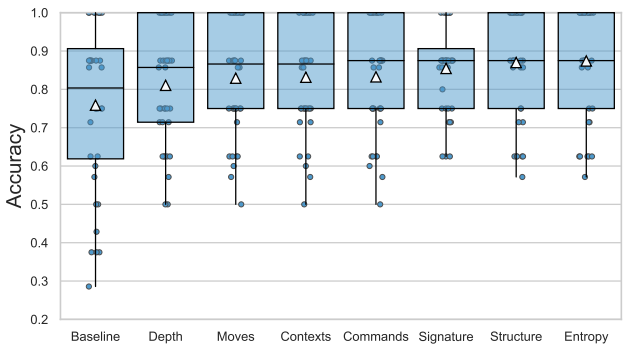

Figure 4: Benchmark evaluation results - individual accuracy values of the complexity metrics as models for a task's difficulty. Triangles denote the mean performance.

Discussion and Conclusion

In this paper we presented seven different complexity metrics which we used in our proposed difficulty modeling task in the algorithmic thinking domain with a focus on a railway environment. We implemented them in a benchmark and evaluated their predictive performance by comparing them against a baseline model and also with their correlation values. We used data from an experiment we conducted whose design is inspired by a previous study in the railway environment by Khemlani et al. (2013).

The best performance achieved by entropy is not a surprising result, as it is expected that a more disorganized

distribution of the wagons on the tracks should lead to more difficulties remembering wagons' positions and adjusting them after performing rearrangement operations. Through the signature metric we also learn that in some cases the immediate repetition of instructions helps individuals when simulating an algorithm.

The structure metric is the second best predictor in our benchmark, even though it doesn't have a significant correlation and it only provides 3 possible values. The combination of a high predictive performance with a relatively low degree of freedom for the threshold indicates that its underlying concept is in fact meaningful. That is in line with the found relevance of lines of code to task difficulty (Sheard et al., 2013; Nguyen et al., 2007).

Analyzing the relation between complexity metrics and perceived difficulty of algorithmic tasks is a topic researched for many years from different perspectives, e.g. understanding mental processes in computational thinking (e.g. Khemlani et al. (2013), education and exam creation (e.g. Sheard et al. (2013)) and software maintenance (e.g. Curtis, Sheppar, Milliman, Borst, and Love (1979)). In such studies, usually a significant correlation is always taken as a sign of a good predictor, but we show that our modeling task gives us the possibility to analyze the complexity metrics' capability to act as predictive models of task difficulty beyond statistical analysis. For example, while the depth metric was considered to be a good predictor based on correlation, it failed to translate to an adequate performance when predicting the complexity for individuals and was outperformed by all other metrics.

Our findings open many doors and possibilities for future, exciting research. In the experiment we found a difference in correctness patterns between 4 and 6 trains. An interesting next step would be to perform further analysis whether this influences the complexity metrics as predictive models for these tasks as well. Additionally, it would be useful to conduct a similar experiment, where individuals are exposed to the same tasks multiple times. That would allow for broadening the modeling task to predictions on new, unseen individual data. Further research steps can be taken by looking deeper into relations between the metrics and examining the predictive powers of their combinations. For example, the contexts metric on its own did not perform as well as the best performing models, but its performance might be bettered by analyzing how switching contexts taken into consideration together with the distribution of wagons (entropy) predicts perceived difficulty. Moreover, it would be of great interest to research whether expanding already existing algorithm-specific complexity metrics towards considering the cognitive load an algorithm can have on an individual would lead to better predictions of difficulty thresholds. Finally, the setting allows for an extended version of the modeling task: Instead of predicting the expected correctness based on complexity, the prediction of the exact responses given by participants to a task can serve as a challenging objective. The extension of the modeling task requires a extensive data set, but would in turn open the task for models that go beyond an estimate of complexity. Instead, models that are able account for and simulate the processes underlying human algorithmic thinking would be required to solve the task.

Acknowledgements

This research was funded by the Saxony State Ministry of Science and Art (SMWK3-7304/35/3-2021/4819) research initiative "Instant Teaming between Humans and Production Systems" on the basis of the budget passed by the deputies of the Saxony parliament.

References

Bertelson, P. (1961). Sequential redundancy and speed in a serial two-choice responding task. *Quarterly Journal of Experimental Psychology*, *13*(2), 90-102.

Bucciarelli, M., Mackiewicz, R., Khemlani, S., & Johnson-Laird, P. N. (2022). The causes of difficulty in children's creation of informal programs. *International Journal of Child-Computer Interaction*, *31*, 100443-100455.

Campbell, D. J. (1988). Task complexity: A review and analysis. *Academy of management review*, *13*(1), 40-52.

Curtis, B., Sheppar, S. B., Milliman, P., Borst, M. A., & Love, T. (1979). Measuring the psychological complexity of software maintenance tasks with the halstead and mccabe metrics. *IEEE Transactions on Software Engineering*, *2*, 96-104.

Effenberger, T., Cechák, J., & Pelánek, R. (2019). Difficulty and complexity of introductory programming problems..

Garavan, H. (1998). Serial attention within working memory. *Memory & Cognition*, *26*(2), 263-276.

Goodwin, G. P., & Johnson-Laird, P. N. (2011). Mental models of boolean concepts. *Cognitive Psychology*, *63*(1), 34-59.

Khemlani, S., Goodwin, G. P., & Johnson-Laird, P. N. (2015). Causal relations from kinematic simulations. In *Proceedings of the 37th annual conference of the cognitive science society* (p. 1075-1080).

Khemlani, S., Mackiewicz, R., Bucciarelli, M., & Johnson-Laird, P. N. (2013). Kinematic mental simulations in abduction and deduction. In *Proceedings of the national academy of sciences* (Vol. 110(42), p. 16766-16771).

Liu, P., & Li, Z. (2012). Task complexity: A review and conceptualization framework. *International Journal of Industrial Ergonomics*, *42*(6), 553-568.

Nguyen, V., Deeds-Rubin, S., Tan, T., & Boehm, B. (2007). A sloc counting standard. *Cocomo II forum*, 1-16.

Riesterer, N., Brand, D., & Ragni, M. (2020). Do models capture individuals? evaluating parameterized models for syllogistic reasoning. In *Proceedings of the 42nd annual conference of the cognitive science society* (p. 3377-3383).

Rogers, H. (1967). *Theory of recursive functions and effective computability*.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379-423.

Sheard, J., Simon, B., Carbone, A., Chinn, D., Clear, T., Corney, M., ... Teague, D. (2013). How difficult are exams? a framework for assessing the complexity of introductory programming exams. In *Proceedings of the fifteenth australasian computing education conference* (p. 145-154).