Empirical Software Engineering (accepted: 2023)

Noname manuscript No.
(will be inserted by the editor)

Do Attention and Memory Explain the Performance of Software Developers?

Valentina Piantadosi · Simone Scalabrino · Alexander Serebrenik · Nicole Novielli · Rocco Oliveto

Received: date / Accepted: date

Abstract Writing and modifying source code are core activities in software development and evolution. The outcome of a coding task in terms of quality may depend on several aspects, such as the difficulty of the task or the complexity of the system. Besides, it is well known that individual characteristics of developers, like the programming experience, play a lead role in this. Recent work started exploring the influence that cognitive human aspects have on the ability of developers to acquire information from the source code (*e.g.*, finding security blind spots). However, it is still unknown to what extent such aspects influence their ability of completing coding tasks.

In this paper, we theorize that two cognitive human aspects, attention and memory, play a role in predicting the outcome of a coding task. We conducted a controlled experiment involving 32 participants (18 bachelor students, 9 master students, 2 Ph.D. students. and 3 practitioners), in which we asked them to complete two bug-fixing and two feature implementation tasks. We measured, for each of them, three attention-related factors (*i.e.*, alerting, orienting, and executive control) and two memory-related ones (*i.e.*, working memory and immediate recall) through well-established psychometric tests. Finally, we investigated to what extent these factors can explain the correctness, the readability and the time taken to complete a task in function of such factors. Our results show that all the attention- and memory-related factors achieved very low correlation with correctness and time. Indeed, the number of years of programming experience is far more important than all the other variables we considered for explaining the correct-

V. Piantadosi and S. Scalabrino and R. Oliveto
University of Molise, Italy
E-mail:
{valentina.piantadosi, simone.scalabrino, rocco.oliveto}@unimol.it

Alexander Serebrenik
Eindhoven University of Technology, The Netherlands
E-mail: a.serebrenik@tue.nl

Nicole Novielli
University of Bari, Italy
E-mail: nicole.novielli@uniba.it

ness and the time required to complete a task. Moreover, we found a significant relationship between orienting (an attention-related factor) and code readability.

Keywords Cognitive Human Factors · Coding Task Outcome Prediction · Empirical Software Engineering

1 Introduction

Software development and evolution are complex activities involving writing and modification of the source code. Developers are continuously presented with coding tasks in their daily activities, such as implementing a new feature or fixing bugs. Being able to estimate the time required to complete a coding task and predict its internal and external quality (*e.g.*, correctness of coding tasks) would allow to better allocate the effort of a development team. Such a problem is not new in the literature. *Defect prediction*, for example, is an active field of software engineering research aiming at inferring if a given software component contains bugs (Zimmermann et al., 2009; He et al., 2012; Rahman et al., 2012; Li et al., 2018; Thota et al., 2020). Similarly, previous work aimed at predicting the time needed to complete tasks, such as fixing bugs (Zhang et al., 2013; Bhattacharya and Neamtiu, 2011; Blackburn et al., 1996).

In the last decade, researchers have started including in such models developer-related factors, such as their attention focus (Posnett et al., 2013) or the scattering of changes they performed (Di Nucci et al., 2017). The majority of existing models for predicting the time and the quality of a coding task, however, only take into account superficial characteristics of developers, and they neglect the role that *cognitive human aspects* can have on the final outcome of a coding task. Cognitive human aspects are measurable aspects of cognitive human functions, such as attention or memory.

The literature in psychology reports that cognitive human aspects correlate with the outcome of several human activities, even complex ones. More specifically, attention- and memory-related factors are shown to be related to activities that require problem-solving. For example, several studies (Weaver et al., 2009; Posner, 1980; Eriksen and Eriksen, 1974) used attention-related factors to predict the outcome in driving, while others have shown that both attention and memory correlate with the performance in mathematics (Musso et al., 2012; Wei et al., 2012; Passolunghi et al., 2007).

In this paper, we theorize that cognitive human aspects, and, specifically, attention- and memory-related ones, play a role in explaining the outcome of coding tasks in terms of *time* required to complete a task, and quality of the final solution, here measured in terms of *correctness* (external quality) and *code readability* (internal quality). As for attention, we focus on three factors: *alerting* (*i.e.*, the ability of gaining and maintaining an alert state), *orienting* (*i.e.*, the ability of selecting information from sensory input), and *executive control* (*i.e.*, the ability of dealing with conflict among responses); as for memory, we consider *immediate recall* (*i.e.*, the ability of recalling information acquired in the short term), and *working memory* (used for elaborating problem-solving strategies (Baddeley, 1983; Shneiderman and Mayer, 1979)). We decided to focus on these cognitive human aspects because Peitek et al. (2018), Siegmund et al. (2014) and Krueger et al.

(2020) observed neural activation related to *attention* and *working memory* during coding activities (*i.e.*, program comprehension and code writing).

Proving our theory would pave the way for different future research directions. First, researchers would be able to define personalized defect prediction models. Second, specific cognitive training sessions (Oded, 2011) for developers could be devised, aimed at improving the most important attention- and memory-related factors to possibly write better code in less time. To test our theory, we conducted a controlled experiment for answering to the following research questions:

- **RQ**$_1$: *To what extent do attention and memory have an impact on the correctness of the solution of coding tasks?*
- **RQ**$_2$: *To what extent do attention and memory have an impact on the time needed to complete coding tasks?*
- **RQ**$_3$: *To what extent do attention and memory have an impact on readability of the solution of coding tasks?*

We involved 32 participants with different backgrounds and asked each of them to complete two bug fixing and two feature implementation tasks. Since we were interested in measuring the influence of cognitive human aspects independently from the context in which such tasks would have been completed (*e.g.*, a specific software system), we provided developers with stand-alone problems (*i.e.*, not requiring the knowledge of other software components) that could be solved in the time we allocated for tasks (30 minutes). We measure attention- and memory-related factors in isolation to collect information about the personal characteristics of developers before of the programming tasks given that cognitive human aspects cannot be measured simultaneously with the programming tasks. We used state-of-the-art psychometric tests to measure attention- and memory-related factors. The main objective of our study is to assess whether there are relations between such variables and three variables related to the outcome of the task: *correctness* (percentage of test cases passed), *time* (minutes required to complete the task) and *readability* (computed using the metric by Scalabrino et al. (2018)).

Our results show that the correlation between all the attention- and memory-related factors and correctness is not statistically significant, both when considered alone and when combined in regression models. The only developer-related variable that we found to be significantly related to *correctness* is *programming experience*. On the other hand, we observe a statistically significant relationship between attention-related factors (alerting, p-value = 0.046 , and orienting, p-value = 0.031) and *code readability*. In addition, there is a significant correlation between a memory-related factor (immediate recall, p-value = 0.013) and *time*. While experiments conducted with different and larger samples are required to further corroborate our findings, our results provide a clear message to future researchers interested in this field: Attention- and memory-related factors should be investigated with other qualities of software (*e.g.*, familiarity, understandability), and to the same way, qualities of software should be investigated consider other cognitive human aspects (*e.g.*, language, intelligence). Also, programming experience should always be taken into account, since it is significantly related to both time and correctness.

The remainder of our paper is organized as follows. In Section 2 we provide the necessary background and we describe the related literature. In Section 3 we present our theory. In Section 4 we report the design of our study, while in Section

5 we show the results of our analysis. Section 6 describes the threats to validity, and Section 7 concludes this paper.

2 Background and Related Work

In this section, we first provide background on psychological notions, such as cognitive functions and aspects. Then, we discuss (i) related work exploring generic applications of the cognitive aspects (*e.g.*, driving and mathematics), and (ii) previous applications of cognitive psychology in the software engineering domain.

2.1 Background

In cognitive psychology, the term 'cognition' refer to the ensemble of thoughts and ideas and is used to denote the internal mental processes (or *cognitive functions*). Cognitive functions regulate human perception, reasoning, memory, intuition, thinking, speaking, decision making, and problem solving (Roy, 2013; Benjafield et al., 2010). Such processes cannot be observed directly but they can be measured indirectly through psychometrics, by using specific tests. Psychometric tests may be, for example, questionnaires or exercises, and they aim at measuring proxy by cognitive functions metrics, such as reaction times, psychological responses, or real-time neuroimaging (Gellman and Turner, 2013; Allan, 2013).

Cognitive functions can be measured through psychometrics tests and neuroimaging. For each cognitive function, there is an appropriate psychometric test. For example, to measure the working memory, it is possible to use the Symbol Digit Modalities Test and the List Sorting Working Memory Test (Gershon et al., 2013). The Brief Test of Adult Cognition by Telephone (BTACT) can be used to measure different aspects of the cognitive functions (*e.g.*, immediate recall) (Tun and Lachman, 2006). Attention level can be measured by the Attention Network Test (ANT)[1] (Posner and Petersen, 1990; Fan et al., 2002; Wang et al., 2004). The proxy variables measured after or during the tests may then be combined to provide a final measure of an *aspect* of the cognitive function (what we call, in this paper, cognitive human aspect). For example, in ANT, reaction times obtained in different situations are combined to measure the *efficiency* of the *alerting*, *orienting*, and *executive control* cognitive functions.

These tests have been used in numerous studies. Zelazo et al. (2013) used ANT to evaluate executive function integrated in the NIH Toolbox Cognition Battery. Brearly et al. (2018) validated the application of these test between NIH Toolbox Cognition Battery on iPad and web platforms. Blank et al. (2020) used it to measure the parenting stress levels with the relationship between parenting stress, language comprehension, and inhibitory control skill in children. Inhibitory control is the inhibition of inappropriate stimuli and response (Coulter, 1983). Tiego et al. (2018) represented inhibitor control as a construct of working memory. Howard et al. (2014) studied the methodology of Friedman and Miyake (2004) evaluating four inhibition models. Finally, Lesage et al. (2020) compared the effect of nicotine dependence and acute nicotinic stimulation. Symbol Digit Modalities Test has been

[1] ANT is also known as "Flanker Inhibitory Control and Attention Test" (*e.g.*, in the NIH Toolbox Cognition Battery))

used as an outcome measure for multiple sclerosis (Benedict et al., 2017; Parmenter et al., 2007; Forn et al., 2013; Genova et al., 2009). In addition, this test has been proved as reliable as fMRI (Functional Magnetic Resonance Imaging) (Forn et al., 2009; Silva et al., 2018) in assessing the working memory. BTACT Word List Recall to evaluate the cognitive functions of chronic dialysis patients (Song et al., 2015), of patients with traumatic brain injuries (Cairncross et al., 2022), and for multiple chronic diseases (Shorey and Friedman, 2018).

A different approach used for studying cognitive functions is through *cognitive neuroscience*. Functional neuroimaging has been used to find correlations between the patterns obtained from the brain activity in various processing stages and different cognitive functions (Roy, 2013).

2.2 The Impact of Cognitive Aspects on Task Performance

Previous work focused on studying cognitive functions in different domains. In some experiments, researchers tried to explain or predict the outcome of non-trivial tasks (*e.g.*, driving). In these studies, the authors investigate the relation between cognitive aspects and task performance of any nature (*i.e.*, linguistic, mathematical and driving).

In **linguistic research**, Nour et al. (2019) analyze the attention network tests in three different groups of participants (*i.e.*, intepreting students, translation students, and professional interpreters). Results show the correlation between two types of language interpretations, *i.e.*, professionals and students, and the attention networks, *i.e.*, alerting, orienting and executive control. Results show that each group of participants has specific attention network dynamics: for example, interpreting students differ from translation students for alertness and executive network. Woumans et al. (2015) studied the relation between language control and non-verbal cognitive control between monolingual, Dutch-French unbalanced bilinguals, balanced bilinguals, and interpreters. Results show that bilinguals can modulate better the nature and extent of a cognitive control advantage compared to interpreters and monolinguals.

Other studies used cognitive human aspects in the **mathematical field** to evaluate whether there is a correlation between cognitive human aspects and elementary mathematics performance (Halberda et al., 2008; Mundy and Gilmore, 2009; Passolunghi et al., 2007). Passolunghi et al. (2007) searched for precursors of mathematics learning in the primary school. Their results show that cognitive abilities (*i.e.*, working memory and counting ability) are associated with mathematics learning. Wei et al. (2012) studied which cognitive human aspects are necessary to obtain advanced mathematics knowledge and skills. Results show that spatial abilities and language comprehension have a strong correlation with performance in advanced mathematics, but the study does not report any significant correlation with numerical processing. Musso et al. (2012) also consider interactions and the influence of cognitive human aspects on mathematical performances. Results show a relevance on educational quality, improvement, and accountability.

Weaver et al. (2009) measure alerting, orienting and executive control efficiency to predict the **driving outcome**. In this research area, the Useful Field of View is used to predict driving performances and it is used to measure the processing speed. For this reason, Weaver et al. (2009) investigated whether the Useful Field

of View is equivalent to the Attention Network Test (ANT), taking a positive result.

Multiple studies demonstrated that the experience in a certain domain can allow the acquisitions of skills for other domains with similar abilities (Green and Bavelier, 2007; Bialystok and DePape, 2009; Habib and Besson, 2009; Schellenberg, 2004) and that a training of some cognitive functions can change attentional processes (Lilienthal et al., 2013).

2.3 Cognitive Human Aspects in Software Engineering

In *Software Engineering research*, previous work measured cognitive human aspects and correlated them with developers' characteristics and software quality. As described in Section 2.1, cognitive human aspects can be measured through neuroimaging and psychometric tests. We describe below previous work in both the areas.

In last decade, neuroimaging techniques have been used to understand the cognitive processes of programming (Ebisch et al., 2013; Floyd et al., 2017; Huang et al., 2019; Peitek et al., 2018; Siegmund et al., 2014, 2017; Krueger et al., 2020; Karas et al., 2021). Specifically, Peitek et al. (2018) and Siegmund et al. (2017) use these techniques in the program comprehension and Krueger et al. (2020) use these techniques in the code writing.

Peitek et al. (2018) analyzed whether functional magnetic resonance imaging (fMRI) can measure program comprehension. The authors invited 17 developers to comprehend source code in a fMRI (functional magnetic resonance imaging) scanner. Results show that five brain regions are activated during a program comprehension task related to working memory, attention level and language processing. The cognitive effort is reduced given developers' familiarity with the programming language. Subsequently, this experiment has been replicated with 11 participants and results have been confirmed (Siegmund et al., 2017). Siegmund et al. (2017) used the fMRI with 11 participants while they read a program. Authors perform manipulations on experimental conditions of beacons and layout to understand cognitive processes of the bottom-up comprehension. Their results show that beacons facilitate program comprehension tasks and there is less brain activation.

Krueger et al. (2020) use functional magnetic resonance imaging (fMRI) to compare neural representation of code writing and those of prose writing. Results show that the prose writing activates the left hemisphere which is associated with language and the code writing involves the activation of more parts of the right hemisphere, *i.e.*, attention control, working memory, planning and spatial cognition. Thus, these results support the evidence that code and prose writing have different behaviour at mental level.

Sharafi et al. (2021) performed two controlled experiments with 112 students during a series of development activities, *i.e.*, code comprehension, code review, and data structure manipulations. During coding activities, students were monitored through neuroimaging activities, *i.e.*, functional near-infrared spectroscopy (fNIRS), functional magnetic resonance imaging (fMRI) and eye tracking. Results show that there are different neural representations between programming languages and natural languages.

There are few previous studies in which psychometric tests were used to predict developers' performance in tasks. Oliveira et al. (2018) performed an experiment with 109 developers in which they studied whether developers can detect API security blindspots in code. An API security blindspots is an error generated from the developer that can conduct to a violation of the API usage (Cappos et al., 2014). In addition, Oliveira *et al.* examined the influence of developer characteristics (*e.g.*, familiarity with code, cognitive human aspects and personality) on the ability in detecting blindspots. Their results showed that there is no correlation between cognitive human aspects and the developers' ability to detect API blindspots. Recently, Brun et al. (2021) replicated the study by Oliveira *et al.* and they obtained similar results. Differently from such studies, in this paper we focus on the relationship between cognitive functions and the outcome of coding tasks, which are inherently different because coding tasks require developers to *write* code, and not just *read* it. In another study, Oliveira et al. (2014) exploit the psychological manipulation to validate the following hypothesis: Software vulnerabilities are not part of classical programming heuristics and developers do not consider them in their programming tasks. Results show that the security is not a principal activity of developers and this task needs of a certain cognitive effort.

3 Cognitive Factors and Software Development Tasks

We hypothesize that attention- and memory-related factors allow to explain the *correctness* of a task, *time* needed to complete it and *code readability* of written code. In the following, we provide more details on our theory.

3.1 Attention

Attention is part of executive functions, as planning, sequencing, and cognitive flexibility (Crawford, 1998). As explained in Section 1, attention can be controlled through three key aspects, *i.e.*, alerting, orienting and executive control, that provide the reactivity to a specific event or stimulus (Posner, 1980).

As reported by Peitek et al. (2018), Siegmund et al. (2017) and Krueger et al. (2020), the attention is a neural activation both for code writing (Krueger et al., 2020) and for program comprehension (Peitek et al., 2018; Siegmund et al., 2017). Thus, attention is an active part of the right hemisphere when a developer writes and comprehends code. Thus, we conjecture that the attention is important both for the implementation of a new feature (code writing) and for fixing of a bug (code comprehension + code writing), and, specifically, we expect that higher efficiency in the *alerting* network would positively affect the *correctness* of coding tasks. We also conjecture that alerting is an important factor that determines the *time* needed to complete a task. If developers have a close deadline, they experience time pressure (Kuutila et al., 2017; Pinto et al., 2017). The influence of time pressure on developers' performance has been studied by Bowrin and King (2010), who show that it results in the introduction of bugs above all if complex tasks are used. Let us imagine that the release deadline is in a hour and that a developer finds a bug. They need to fix it in the minimum time possible. To do this, the developer cannot be distracted and they need to maintain an alert state.

We expect that *orienting* affects the way developers apply coding conventions in their writing activities and verify whether their own code is readable or not since they would be more prone to focus on specific areas of the code. Finally, a higher efficiency in the *executive control* network would allow developers to complete tasks better (*correctness*), more quickly (*time*) and to implement readable code (*readability*) because an individual with a high executive control is able to better isolate conflicting stimuli (*e.g.*, the ones from the environment from the ones of the task) and, thus, focus more on the task at hand.

3.2 Memory

In the studies of Peitek et al. (2018), Siegmund et al. (2014), and Krueger et al. (2020), memory is another neural activation during code writing and the program comprehension. The *immediate recall* is part of the episodic memory and this memory contains a good amount of past events (Schacter et al., 2000). There are three reasons why memory is important: ability to recall, ability to elaborate solutions and ability to manipulate numbers and words. In the case of coding tasks, developers with a good ability to recall might be able to better re-use patterns of solutions applied in the past in similar situations. For example, developers have to fix a bug and they have fixed a similar bug in past; thus, developers could remember the related fix. This would result in saving time and, possibly, in higher chances of spending more time ensuring that the solution provided is correct (*i.e.*, in higher *correctness*) and also of writing readable code (*i.e.*, high *readability*). In addition, the human *working memory* provides a temporary storage of information necessary for other cognitive tasks (*e.g.*, reading or problem-solving) (Baddeley, 1983). Specifically, developers might use their working memory to elaborate solution strategies (Shneiderman and Mayer, 1979). Furthermore, developers with a strong working memory can also manipulate numbers and words to write readable code (Peitek et al., 2018). We conjecture that a good working memory allows developers to complete a coding task both more correctly and more efficiently (*i.e.*, shorter *time* needed). We choose the BTCAT Word List Recall and the Symbol Digit Modalities Test because these two tests were used in a similar study (Oliveira et al., 2018): Oliveira *et al.* measured the influence of developer characteristics on the ability in detecting blindspots. In addition, factors of attention and of memory are specific to tasks related to reasoning, memory and problem solving (Roy, 2013; Benjafield et al., 2010).

4 Design of the Study

The *goal* of the study is to verify to what extent attention and memory have an impact on how developers complete coding tasks. The *perspective* is of researchers that aim at measuring the influence of cognitive human factors on developers' performance on the (*time* needed to complete a task, *correctness* and *readability* of the solution).

Specifically, we formulate and address the following research questions:

- **RQ**$_1$: *To what extent do attention and memory have an impact on the correctness of the solution of coding tasks?*

- **RQ**$_2$: *To what extent do attention and memory have an impact on the time needed to complete coding tasks?*
- **RQ**$_3$: *To what extent do attention and memory have an impact the readability of the solution of coding tasks?*

To answer our research questions, we conducted a controlled experiment in which we collect measurements for both the dependent (*i.e.*, *correctness*, *time*, and *readability*) and the independent variables (attention- and memory- related factors). Besides such factors, we also include other control variables that are commonly associated with the outcome of coding tasks, *i.e.*, *task type*, *task difficulty*, and developer's *programming experience* (Juristo and Moreno, 2013).

Task Type. Some developers might find easier to implement a new feature from scratch, because they do not need to deal with code written by other developers; some others, instead, might find it easier to start from a partial solution and modify it to make it work as intended. Therefore, task type may be naturally associated with the *correctness* of the solution. We also assume that task type explains variance in the *time* needed to complete a task: we expect that *bug fixing* tasks generally require more time since most of the code is written; however, this might strongly depend on the developer, as previously explained by Rasch and Tosi (1992).

Task Difficulty. Difficulty is naturally associated both with the *correctness* of the solution: intuitively, it is more likely that a developer writes bug-free code when presented with an easy task; similarly, we can assume that easier tasks take generally less time to be completed. While difficulty is somewhat a subjective concept and it might depend on the knowledge and experience of the developer, there are some objective features that make some tasks more difficult than others. For example, everything else being equal, fixing a bug that involves one line of code is inherently easier than fixing a bug involving multiple lines of code (Rasch and Tosi, 1992).

Experience. It is a common understanding that the *programming experience* plays a significant role in different kinds of software engineering tasks (Ricca et al., 2007, 2009; Siegmund et al., 2014). We assume that more experienced developers can complete tasks more correctly (higher *correctness*) and more quickly (lower *time*). We measure the programming experience by asking developers to report the number of years of experience (*i.e.*, from their first programming task).

In the following, we describe (i) the context of our experiment, *i.e.*, the participants and the tasks, (ii) the procedure we used to collect the data, *i.e.*, how we run the experiment, and (iii) how we analyzed the collected data to answer our research questions.

4.1 Context Selection

The context of our study is composed of *objects*, *i.e.*, coding tasks, and *subjects*, *i.e.*, software developers. To select the tasks we used in our study, we relied on LeetCode,[2] an online platform commonly used to exercise coding problems. The platform proposes a wide range of problems that can be solved by users in many programming languages. A problem in LeetCode is usually composed by (i) a

[2] https://leetcode.com/

```
public boolean match(String word, String pattern) {
  Map<Character, Character> m1 = new HashMap();
  Map<Character, Character> m2 = new HashMap();

  for (int i = 0; i < word.length(); ++i) {
    char w = word.charAt(i);
    char p = pattern.charAt(i);
    if (!m1.containsKey(w)) m1.put(w, p);
    if (!m2m1.containsKey(p)) m2m1.put(p, w);
    if (m1.get(w) != p || m2m1.get(p) != w)
      return false;
  }

  return true;
}
```

Fig. 1: Example of an injected bug in the easy task. We decide to remove the hash map m2 from the originally correct solution.

description of the problem, and (ii) at least an example of input and expected output. The developer can, then, implement a solution, manually test it, or submit it. In the latter case, LeetCode runs a test suite to check if the solution is correct. To select the participants, instead, we used convenience sampling, and we invited people within the personal network of the authors and through student channels. We provide below more details about the selection of tasks and participants.

Task Selection. The two aspects we controlled for in our experiment in terms of task are the difficulty and the task type. As for the difficulty, we aimed at having both *easy* and *hard* tasks. As for the task type, we wanted to cover two categories of tasks typically performed during software development and maintenance: *feature implementation* and *bug fixing*. In the first category, developers are requested to implement code from scratch, while, in the second one, they are provided with a partially correct solution that they need to modify. We defined a total of 4 tasks to cover all the possible combinations of task difficulty and type.

To define the tasks, as a first step, we started from the pool of all the problems available in LeetCode, and filtered them based on the difficulty tags. The difficulty tag on LeetCode is manually assigned by the person who originally proposed the problem. We selected two separated pools of problems: *easy* ones and *hard* ones, discarding problems of *medium* difficulty. We arbitrarily picked two easy tasks and two hard tasks from such pools. We verified that for all the problems we selected it was possible to submit a solution in Java since we planned to ask developers to complete the tasks using Java.

The definition of the *feature implementation* tasks starting from the LeetCode problems was straightforward: We simply provided the participants with the problem description and we asked them to implement a solution from scratch. To define the *bug fixing* tasks, instead, we needed to provide a buggy solution that they would have fixed. To define the buggy solution, we started from the correct solution provided by LeetCode itself. Then, we manually injected bugs in such a solution. The type of modification we made to the code depended on the difficulty of the task (*i.e.*, we introduced a more articulated bug in the hard task). We report an example of an injected bug in Figure 1. We report in Table 1 the main features

Table 1: Tasks selected for the experiment. 🐞 and 💧 indicate task types (*bug fixing* and *feature implementation*), while ⏬ and ⏫ indicate task difficulty (*easy* and *hard*).

Problem	Type	Difficulty	#Test Cases
Find and Replace Pattern (LeetCode, 2020b)	🐞	⏬	47
Sort the Matrix Diagonally (LeetCode, 2020d)	🐞	⏫	15
Duplicate Zeros (LeetCode, 2020a)	💧	⏬	30
Regular Expression Matching (LeetCode, 2020c)	💧	⏫	352

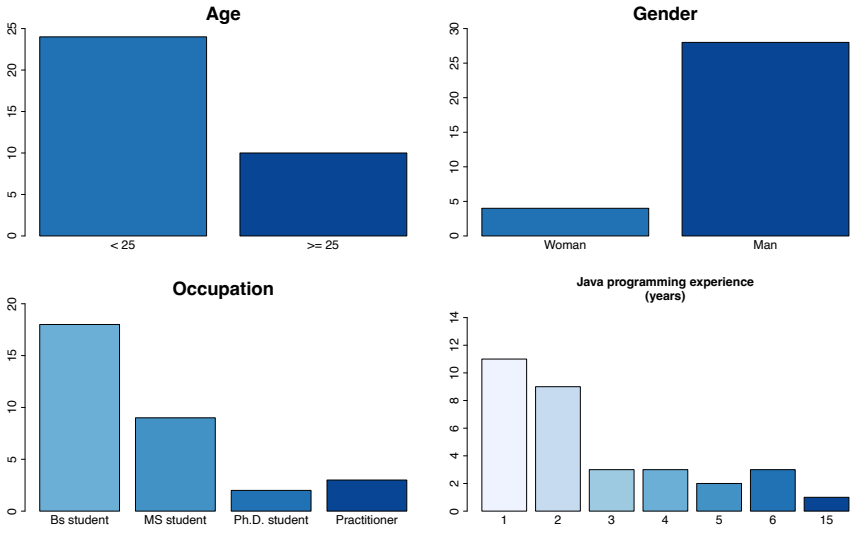

Fig. 2: Demographic information about the participants.

of the four tasks we defined. Specifically, for each task we report the difficulty level of tasks, *i.e.*, the value reported when we selected the task. The difficulty of both the bug fixing tasks was changed to "Medium" in LeetCode after we selected the tasks and run the experiments. In addition, for each task we report the number of test cases on LeetCode. Participants did not have access to these test cases. We internally used them to check the *correctness* of the solutions they provided.

Participants Selection. To define the number of data points we would have needed to observe variations in the dependent variables due to the factors we studied, we ran a power analysis for linear regression.[3] To identify a model with $f^2 \simeq 0.15$ (*i.e.*, $R^2 \simeq 0.13$) using 8 predictors (more on the model in Section 4.3) with a 80% power, we needed at least 109 data points. Since each participant produces four data points, one for each task, we needed at least 28 participants. As output of the recruiting phase, we involved 32 participants, distributed as follows: 18 bachelor students, 9 master students, 2 Ph.D. students, and 3 practitioners.

[3] We used the tool available at `https://www.statskingdom.com/sample_size_regression.html`

Figure 2 reports some demographic information about the participants we selected.

4.2 Data Collection

To collect data through our controlled experiment, the first step consisted in (i) selecting the psychometric tests we should have used to measure the attention- and memory-related factors we were interested in, and (ii) implementing them in a web-app that allowed us to administer such tests. The second step was to conduct the experiment. We describe below in detail these two steps.

4.2.1 Psychometric Tests

We created a web-app that contains all psychometric tests we were interested in. Through this web-app, we administered psychometric tests to participants and we could automatically obtain their results. To measure the attention-related factors, we used ANT (Fan et al., 2002). In such a test, the participant is presented with five arrows, each one pointing either to the left or to the right. The goal of the participant is to indicate the direction of the central arrow as quickly as possible. Such a procedure is repeated several times and the arrow sets are presented in different ways. Some of them are preceded by a cue, *i.e.*, a hint about the time (*double cue*) and/or the place (*spatial cue* or *central cue*) in which the arrows will appear in the next few instants . Some others, instead, are presented without any cue (*i.e.*, they are just shown on the screen). When a cue is given, sometimes it is coherent with the time/location in which the arrows will appear, sometimes, instead, it is conflicting (*e.g.*, the cue is shown at the top, but the arrows appear at the bottom). It is possible to watch the video of the test in our replication package (Piantadosi et al., 2021). The web-app measures the response time (RT), *i.e.*, the time the participant takes to select answer. Besides, the web-app annotates, for each RT, the type of cue and whether it was conflicting or not. In total, we aimed at collecting 128 RT measures for each participant; the test lasted about 10 minutes. After the first half (64 evaluations), participants could pause for as long as they wanted before continuing with the second half. We used the 128 RT measurements to compute the *alerting*, *orienting*, and *executive control* efficiency metrics through the following formulas (Fan et al., 2002):

$$\begin{aligned} \textit{alerting} &= \textit{mean}(RT_{\text{no cue}}) - \textit{mean}(RT_{\text{double cue}}) \\ \textit{orienting} &= \textit{mean}(RT_{\text{central cue}}) - \textit{mean}(RT_{\text{spatial cue}}) \\ \textit{executive control} &= \textit{mean}(RT_{\text{coherent}}) - \textit{mean}(RT_{\text{conflicting}}) \end{aligned}$$

The values of the metrics provide with the efficiency of the *alerting*, *orienting*, and *executive control* networks of the participants, respectively. Higher values generally indicate that a given type of cue is effective and, therefore, the network (*i.e.*, alerting) is more efficient. For example, for *orienting*, a value greater than 0 indicates that the spatial cue allows the participants have better response times, *i.e.*, they are able to focus on the cued area. The range of response times is between 0 and 1700 ms.

The main test is preceded by a tutorial version of the test that lasts ∼3 minutes, in which we allow the participants to familiarize with the web-app.

To measure the memory-related factors, we use two tests: the BTACT Word List Recall test (Tun and Lachman, 2006) for the *immediate recall* and the Symbol Digit Modalities Test (Fellows and Schmitter-Edgecombe, 2020) for the *working memory*. The *BTACT Word List Recall* is part of the BTACT battery of cognitive processing tasks for adults. It allows to measure the *immediate recall*, or, more precisely, the immediate episodic memory for verbal material. The participants were asked to carefully listen to a set of 15 registered words, that we call C. Then, they were asked to repeat all the words they could remember in 90 seconds. Participants could use a button to indicate that they could not remember other words to proceed with the test. To assign a score, one of the authors listened to each recording offline, and manually annotated the words pronounced by each participants. Since most of the participants were non-native English speakers, we were tolerant for imperfect pronunciations, as long as it was clear that they referred to a word in the list C. This is the main reason why we did not use automated speech recognition.

For each participant, given the list of words pronounced, P, we compute the *immediate recall* as $|P \cap C|$, *i.e.*, the number of words correctly recalled. The *Symbol Digit Modalities Test* allows us to measure processing speed and working memory. In this test, participants were presented with a coding key mapping nine abstract symbols to numbers from 1 to 9. Participants memorize the mapping for the time they need. Then, they were presented 144 symbols in a random order with the mapping still being visible. Finally, the participants had 120 seconds to "decode", *i.e.*, to write in numbers, as many symbols as possible in the exact order they were presented. Participants could not skip symbols. We compute the *working memory* score as the total number of correctly decoded symbols.

4.2.2 Controlled Experiment Protocol

Before starting the experiment, we asked each participant to fill in a form through which they provided basic demographic information, *i.e.*, gender, education level, occupation, and years of experience, both with the Java programming language and overall. The experiment was divided, for each participant, in two sessions, held on different days, behaving risks and benefits. We risked that developers could refuse the participation in part of the experiment (*e.g.*, second session). From the positive side, participants could not get too tired because we decrease the number of tasks for day. The alternative was to have the entire experiment on a single day. Using this alternative, we risked that developers refused the invitation to the experiment for the prohibitive experiment duration. Another risk with this alternative would have been that we would have had a much stronger tiring effect on participants. Each session had the same structure. During each session, the first step was to administer the three previously described psychometric tests in the following order: BTACT Word List Recall (for *immediate recall*), Symbol Digit Modalities (for *working memory*), and, finally, ANT (for *alerting*, *orienting*, and *executive control*). We measured memory-related factors at the beginning of the experiment because there could be the risk that participants were tired after being administered with ANT. These test have to be administered immediately before software development tasks because attention- and memory-related factors have

a short-lived validity, as demonstrated in several previous studies (Rapport et al., 2009; Koen et al., 2013; Conway et al., 2008). We choose to measure psychometric tests both on day 1 and day 2 because psychometric tests used to assess cognitive human aspects have a short validity in time (Hughes, 2018). The second step consisted in asking the participants to complete two of the four programming tasks outlined previously. In this phase, the author, who supervised the experiment, shared a Java file containing (i) the description of the task (*i.e.*, the problem description from LeetCode) and (ii) either a partial solution (for bug fixing tasks) or the boilerplate for implementing the solution, such as the definition of the class and the method that LeetCode expected (for feature implementation tasks). Each participant had 30 minutes to complete each task. After 30 minutes, the author who supervised the experiment asked the participants to submit remotely the solution implemented.

To avoid biases due to the task execution order, we divided the participants in four groups. Depending on the group, a given participant was assigned the tasks in a different order and in different sessions. We defined the groups so as to assign one easy and one hard task, as well as a *bug fixing* and a *feature implementation* task for each session. This allowed us to control for fatigue (*e.g.*, the second task of the day could be performed systematically worse than the first one because the participant was tired) and learning (*e.g.*, participants could get quicker at completing coding tasks in the second session because they trained in the first one). We report the order in which the tasks were assigned to each group in Table 2. Because of the COVID-19 pandemic, it was not possible to completely control for the environment in which the tasks were performed (*e.g.*, conduct the study in a laboratory with the same equipment). Hence, we performed the study adopting a remote setting, trying to recreate the lab setting we originally designed for the lab study. Specifically, each execution session was remotely supervised by the first author and, at each time, no more than two participants worked at the same time. Furthermore, the author who controlled the experiment asked participants to turn on the microphone and the webcam, and to share the screen, so that it was possible to verify the presence of any distractions. The author who attend the experiment both guided the participants through the psychometric tests and the tasks, and ensured that the tasks were performed as they were intended (*e.g.*, ensured that the solution in the determined time frame, control whether there were external distractions, such as phone calls). The psychometric tests were always administered to one participant at a time, to avoid the risk of participants affecting each others behavior (*e.g.*, hearing the words in the Symbol Digit Modalities Test). The participants were not aware of the fact that the tasks were created using LeetCode problems. The author who attended the experiment made sure that the participants did not consult solutions found on the web. Participants shared their screens during the entire experimental sessions.

We operationalize the participants' performance in terms of *time* required to complete the task and *quality of the solution* (*correctness* and *readability*). We use *time*, *correctness* and *readability* as dependent variables in the statistical analysis in Section 4.3. To measure *time*, the author who supervise the experiment manually recorded the time at which each participant started each task and the time at which they reported that they concluded the task. We measure the time in minutes.

To measure the *correctness*, we relied on the test suite provided by LeetCode, as shown in Table 1. One of the authors copied and ran each solution given by the

participants in LeetCode and measured the number of passed test cases for the task x, T_x^+. For feature implementation tasks, we compute the *correctness* simply as:

$$Correctness_x^{FI} = \frac{T_x^+}{T_x},$$

where T_x is the total number of test cases run by LeetCode for x. For bug fixing tasks, instead, we could not use the same formula: The two bug fixing tasks already came with a wrong solution. If developers left the provided solutions, without modifying them they would have achieved higher *correctness* on the task with the less buggy solution (the two buggy solutions had a different starting *correctness*). To avoid this, we measured, instead, the relative change in correctness: Ideally, a participant should achieve 100% *correctness* if they achieve 100% of passed tests, 0% if the *correctness* does not change compared to the initial solution, and -100% if no tests pass. To achieve this, we used the following formula:

$$Correctness_x^{BF} = \begin{cases} \frac{T_x^+ - BT_x^+}{1 - BT_x^+}, \text{ if } T_x^+ \geq BT_x^+ \\ \frac{T_x^+ - BT_x^+}{BT_x^+}, \qquad \text{otherwise} \end{cases}$$

where BT_x^+ is the number of tests passed with the buggy solution provided by the experimenters.

To measure *readability*, we used the approach defined by Scalabrino et al. (2018). While the model returns a binary assessment (*i.e.*, *readable* or *unreadable*), it also provides a number which ranges between 0 and 1, which represents the estimated probability that the given snippet is readable. We use such a continuous value in this study. In this case, however, we only consider feature implementation tasks. Indeed, the readability of the solutions of bug-fixing tasks might strongly depend on the partial solution we provided and we did not explicitly ask developers to improve the readability of the provided solution. While we did not ask developers to write readable code in feature implementation tasks as well, it is worth noting that writing code from scratch forces developers to make decisions that affect readability (*e.g.*, deciding identifiers' names).

Before running the experiment, we obtained the approval of the ethical board of our research institutions (ID number: ERB2021MCS5). Also, we ran a small pilot study with three additional participants (not involved in the main study), in order to test the web-app and the protocol and to spot any possible problem before starting the study. Participants to the pilot study declared that they had no problems in performing the study on two different days.

4.3 Data Analysis

To answer both our research questions, we initially compute the correlation between each continuous independent variable and the two dependent variables, to understand if there is any direct relationship between couples of dependent/independent variables. To do this, we use the Spearman rank correlation ρ (Spearman, 1961). For the Spearman rank correlation, the correlation is **weak** if the coefficient is between -0.3 and +0.3, **moderate** if the coefficient is between -0.3 and -0.7 or

Table 2: Task assignment for each group. 🐞 and 🥚 indicate task types (*bug fixing* and *feature implementation*), while ⌄⌄ and ⌃⌃ indicate task difficulty (*easy* and *hard*).

Group	Session 1		Session 2	
	1st Task	2nd Task	1st Task	2nd Task
1	🐞⌄⌄	🥚⌃⌃	🥚⌄⌄	🐞⌃⌃
2	🥚⌄⌄	🐞⌃⌃	🐞⌄⌄	🥚⌃⌃
3	🐞⌃⌃	🥚⌄⌄	🥚⌃⌃	🐞⌄⌄
4	🥚⌃⌃	🐞⌄⌄	🐞⌃⌃	🥚⌄⌄

between +0.3 and +0.7, **strong** if the coefficient is less than -0.7 or greater than +0.7 (Sloan, 2015).

Then, to also account for interactions between independent variables, we combine them using explanatory regression models. Specifically, we use generalized linear regression models with Gaussian link function. The independent variables we use for such models are *alerting*, *orienting*, *executive control*, *working memory*, *immediate recall*, *Java programming experience*, *task difficulty*, and *task type*. To answer RQ_1, RQ_2, and RQ_3 we use *correctness*, *time*, and *readability* as the dependent variables, respectively. Therefore, we define three models, one for each dependent variable. For each model, we report its explanatory power (R^2 and R^2_m), the AIC, and the significance obtained for each independent variable (p-values), to understand to what extent they explain *time*, *correctness*, or *readability*. If one of the variables obtains a p-value lower than 0.05, we reject the null hypothesis that such a variable does not explain the dependent variable. Additionally, we use backward stepwise elimination to gradually remove independent variables that give a non-significant benefit to the model and obtain a minimal model for explaining both our dependent variables. To this aim, we start with a full model containing all the independent variables, M_0. Then, we progressively remove the independent variable which is less likely to have a relationship with the dependent variable (*i.e.*, the one with the highest p-value), we define a new model, M_1, and we measure its AIC. We repeat this steps, until the AIC of the model M_{i+1} is lower than the one of the previous version, M_i; in that case, we keep M_i as the minimal model (Efroymson, 1960).

Finally, to corroborate our findings, for RQ_1 we check if there is any significant difference in the independent variables between tasks correctly completed (*i.e.*, 100% correctness) and tasks with at least one failing test case (*i.e.*, correctness lower than 100%). We carry out a similar analysis for RQ_2: In this case, we check the difference between tasks completed before the time was up (*i.e.*, in less than 30 minutes) and when the time was over (*i.e.*, exactly 30 minutes). For RQ_3, we check if there is any significant difference between readable and unreadable solutions, based on the binary classification provided by the readability model. In both the cases, we adopt two hypothesis tests, depending on the variable type: We use the Mann-Whitney U test (Mann and Whitney, 1947) for continuous variables such as time and readability, while we use the Fisher exact test (Fisher, 1922) for categorical variables such as correctness. In RQ_1, the null hypotheses are: *"There is no difference in the independent variable x between tasks correctly completed and tasks with at least a bug"*; in RQ_2, the null hypotheses are: *"There is no difference in the independent variable x between tasks completed before the time was up and*

tasks completed when the time was over"; in RQ_3, the null hypotheses are: *"There is no difference in the independent variable x between readable solutions and unreadable solutions"*. For each RQ, we adjust the p-values for multiple comparisons using the Benjamini and Hochberg procedure (Benjamini and Hochberg, 1995). We also report the effect size, using the Cliff's delta (Cliff, 1993), to understand the magnitude of differences observed. Cliff's delta δ lays in the interval [-1, 1]: the effect size is **negligible** for $|\delta| < 0.148$, **small** for $0.148 \leq |\delta| < 0.33$, **medium** for $0.33 \leq |\delta| < 0.474$, and **large** for $|\delta| \geq 0.474$ (Cliff, 1993). If $\delta > 0$, it means that the first distribution is larger than the second one, while the opposite happens otherwise (Cliff, 1993). To do this, we use Spearman rank correlation ρ (Spearman, 1961) because this is nonparametric, *i.e.*, they do not assume that data follow a specific underlying distribution. Same motivation is for Mann-Withney U test and Fisher exact test: these test are nonparametric.

4.4 Replication Package

All the anonymized data acquired in the experiment are available in our replication package (Piantadosi et al., 2021), which also includes the four tasks (*i.e.*, description and solution of tasks), the script used for statistical analysis and the source code of the webapp.

5 Results

In this section, we provide empirical evidence to answer our research questions and discuss our results in Section 5.4. Overall, the participants achieved 26% of average *correctness* (21% for *bug fixing* and 32% for *feature implementation* tasks). 25% of the participants were able to achieve 100% correctness (30% for *bug fixing* and 22% for *feature implementation*). On average, the participants involved in our study completed the tasks in ∼25.5 minutes (25.4 for *bug fixing* and 25.6 for *feature implementation* tasks). 59% of the participants completed the task in 30 minutes, *i.e.*, they submitted when the time was over (58% for *bug fixing* and 61% for *feature implementation* tasks). Finally, 96.9% of the solutions were readable in feature implementation tasks. In Figure 4 we show the distribution of readable and unreadable code. Figure 3 plots the pairwise relationships between each independent variable and the three dependent ones we investigate, *i.e.*, *correctness* (RQ_1), *time* (RQ_2), and *readability* (RQ_3). We use scatter plots for continuous variables and box plots for categorical (binary) ones. Before commenting this figure, as mentioned in Section 4, it is worth noting that some values of correctness could be between 0 and -1. Specifically, these values correspond to the correctness of bug fixing tasks. As described in Section 4, the correctness of bug fixing tasks could be equal to -100% when no test case passes on the final solution (*i.e.*, not even the ones that passed on the partial solution provided). For *readability*, we do not report the relationship with the *task type* variable since we only considered feature implementation tasks, as explained in the design. The first insight we get from such a figure is that there is no clear relationship between pairs of independent and dependent variables. Interestingly, only a small difference in *correctness* and *time* can be noticed for task-related variables (*i.e.*, *difficulty* and *type*). It is

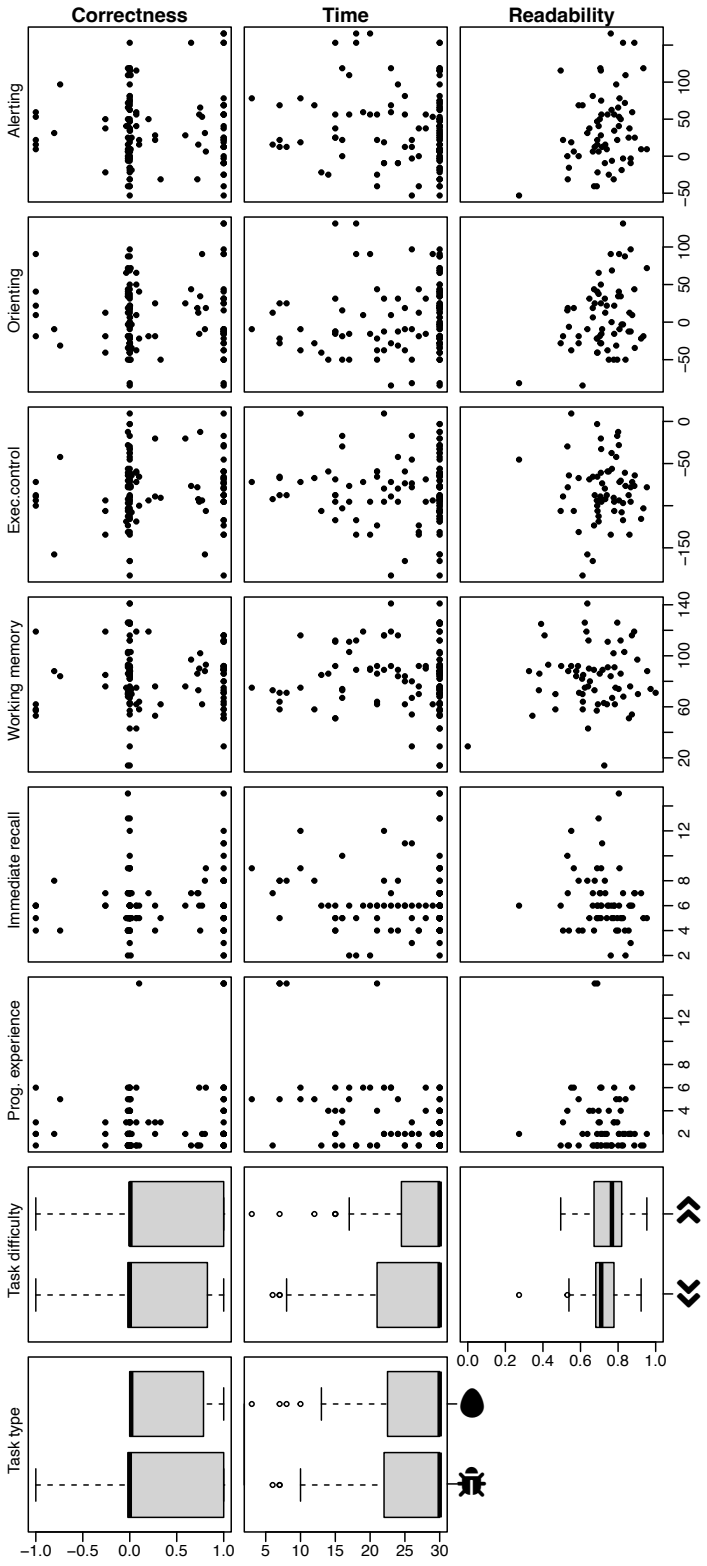

Fig. 3: Relationships between the independent variables (y axis) and dependent variables (x axis). We use scatter plots for continuous variables and box plots for binary ones. 🐞 and 💧 indicate task type (*bug fixing* and *feature implementation*), while ⏬ and ⏫ indicate the task difficulty (*easy* and *hard*).

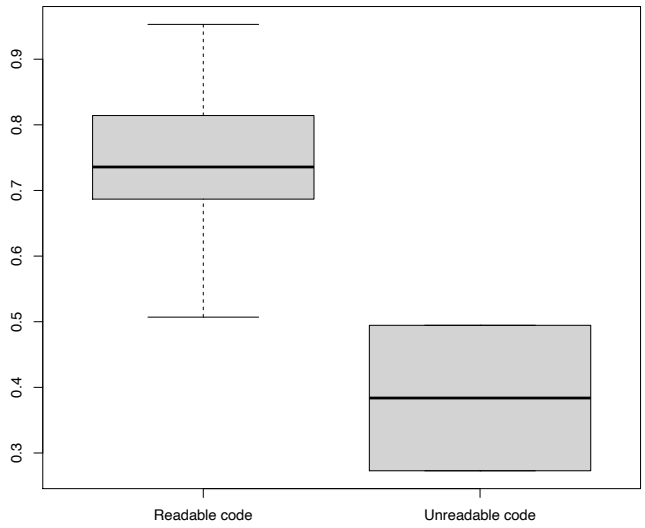

Fig. 4: Boxplot of readability values related to feature implementation tasks.

Table 3: Spearman rank correlations ρ between independent variables and dependent ones (significant correlations in bold).

	Full dataset		**FI dataset**
	Correctness	**Time**	**Readability**
Alerting	-0.006	-0.106	**0.234**
Orienting	-0.041	0.160	**0.208**
Executive control	0.076	-0.020	0.016
Working memory	0.009	-0.032	0.005
Immediate recall	0.086	-0.017	-0.189
Programming experience	0.164	**-0.324**	-0.073

also possible to notice a slight correlation between readability and two attention-related factors, *i.e.*, orienting and alerting: Higher attention seem to be associated to higher readability scores. This visual intuition is later quantitatively confirmed in the results of RQ_3.

5.1 RQ_1: Impact of Attention and Memory on Correctness

Table 3 reports the Spearman ρ correlation coefficients between each independent variable and *correctness*. The first clear fact that can be deduced from the correlations is that both attention-related factors (*alerting*, *orienting*, and *executive control*) and memory-related ones (*immediate recall* and *working memory*)

Table 4: Explanatory models for *correctness*, *time* and *readability*. For minimal models, we report the step of backward stepwise elimination at which each marginally relevant independent variable was removed. For each model, we also report AIC, R^2, and R^2_m.

Variable	**Correctness**			
	Full Model		**Minimal Model**	
	Coefficient	***p*-value**	**Coefficient**	***p*-value**
Intercept	-3.563e-02	0.882	0.15104	**0.028**
Alerting	8.347e-05	0.936	*Removed at step 2*	
Orienting	5.931e-04	0.598	*Removed at step 4*	
Executive control	1.332e-04	0.922	*Removed at step 3*	
Immediate recall	1.900e-02	0.389	*Removed at step 6*	
Working memory	-1.671e-04	0.939	*Removed at step 1*	
Prog. experience	3.707e-02	**0.040**	0.03816	**0.027**
Task type	1.183e-01	0.210	*Removed at step 7*	
Task difficulty	-5.998e-02	0.524	*Removed at step 5*	
	AIC: 212, $\mathbf{R^2}$: 0.06, $\mathbf{R^2_m}$: < 0.01		**AIC**: 201, $\mathbf{R^2}$: 0.03, $\mathbf{R^2_m}$: 0.03	
Variable	**Time**			
	Full Model		**Minimal Model**	
	Coefficient	***p*-value**	**Coefficient**	***p*-value**
Intercept	29.155228	< **0.001**	29.1137	< **0.001**
Alerting	-0.022749	0.056	*Removed at step 7*	
Orienting	0.010888	0.398	*Removed at step 4*	
Executive control	0.005244	0.735	*Removed at step 3*	
Immediate recall	-0.073613	0.770	*Removed at step 2*	
Working memory	0.028154	0.260	*Removed at step 5*	
Prog. experience	-1.246597	< **0.001**	-1.2407	< **0.001**
Task type	-1.542033	0.153	*Removed at step 1*	
Task difficulty	0.154936	0.885	*Removed at step 6*	
	AIC: 836, $\mathbf{R^2}$: 0.28, $\mathbf{R^2_m}$: 0.23		**AIC**: 828, $\mathbf{R^2}$: 0.23, $\mathbf{R^2_m}$: 0.24	
Variable	**Readability**			
	Full Model		**Minimal Model**	
	Coefficient	***p*-value**	**Coefficient**	***p*-value**
Intercept	0.722830	< **0.001**	0.722350	< **0.001**
Alerting	0.000439	0.182	*Removed at step 6*	
Orienting	0.000717	**0.048**	0.000876	**0.012**
Executive control	0.000226	0.597	*Removed at step 2*	
Immediate recall	-0.007807	0.264	*Removed at step 5*	
Working memory	0.000587	0.396	*Removed at step 4*	
Prog. experience	-0.002530	0.655	*Removed at step 1*	
Task type	//	//	//	//
Task difficulty	0.020942	0.486	*Removed at step 3*	
	AIC: -82, $\mathbf{R^2}$: 0.18, $\mathbf{R^2_m}$: 0.08		**AIC**: -88, $\mathbf{R^2}$: 0.10, $\mathbf{R^2_m}$: 0.08	

achieve very low correlations with correctness. In particular, the highest correlation is observed for *immediate recall* (∼0.09). The *programming experience* achieve the highest correlation ($\rho \simeq 0.16$). No correlation, however, is significant, when the p-values are adjusted with the Benjamini and Hochberg procedure (Benjamini and Hochberg, 1995). In absolute terms, all the correlations with single independent variables are very low.

We combine such variables using a generalized linear model, and we report in Table 4 (upper part), for each independent variable, the coefficient and the p-value obtained. The model confirms what the individual correlations suggested: The *programming experience* is the only important variable (p-value = 0.040). It is interesting to note that the *programming experience* in the full model is significant also putting it in relation with other factors. In addition, it is the only variable that remains in the minimal model after applying backward stepwise elimination. In such a model, it achieves, again, statistical significance (p-value = 0.027). The resulting model, however, has a very low explanatory power ($R^2_m = 0.03$ for the minimal model). Also in this case, no attention-related and memory-related factor is significant, and the only factor that remains after backward stepwise elimination is the *programming experience*.

Finally, Table 5 reports the results of the comparisons between distributions of independent variables in the two groups taken into account (*correct* and *incorrect*). While, again, there is no significant difference, we observed a non-negligible (*small*) effect size for programming experience.

Summary of RQ$_1$. Attention- and memory-related factors do not correlate with the correctness of a task. Conversely, programming experience shows a statistically significant correlation with correctness, albeit with small effect size.

5.2 RQ$_2$: Impact of Attention and Memory on Time

As previously done for RQ$_1$, we compute the Spearman ρ correlation coefficients between the independent variables and *time*. We report the results in Table 3. We observe that, while the correlations are low, they are generally higher than the ones achieved for *correctness*. In this case, one of the attention-related variables, *i.e.*, *orienting*, achieves a relatively higher correlation coefficient ($\rho \simeq 0.16$). Such a correlation, however, is not significant, as all the ones with all the other independent variables except for one, *i.e.*, *programming experience*. Such a correlation is high compared to the others ($\rho \simeq 0.32$), but it is still weak in absolute terms. With this correlation, we can state that participants having more experience tend to complete coding tasks faster.

We report the generalized linear model for *time* in Table 4 (middle part). *Programming experience* appears to be, also for *time*, the most important factor (p-value < 0.001). It is worth noting that one of the attention-related factors, *i.e.*, *alerting*, is almost significant (p-value = 0.056) in the full model. However, such a factor is not selected in the minimal model. *Programming experience* is confirmed to be, again, the only relevant factor. For *time*, the minimal model achieves a much higher explanatory power than the one we built for *correctness* ($R^2_m = 0.24$).

Finally, we report in Table 5 the results of the comparisons between distributions of independent variables in the groups *time-up* (task finished in exactly 30 minutes) and *non-time-up* (task finished before the time was over). In this case, we have two variables for which the difference is non-negligible (*small*) in terms of effect size, *i.e.*, *orienting* and *programming experience*. The last one, in particular, is significantly higher for the coding tasks completed before the time was over, as expected.

Summary of RQ$_2$. *Alerting* and *orienting* are attention-related metrics that show a significant correlation with the time needed to complete a task.

5.3 RQ$_3$: Impact of Attention and Memory on Code Readability

As discussed in RQ$_1$ and RQ$_2$, in Table 3 we report the Spearman ρ correlation coefficients between our independent variables and *readability*. Differently from the previous research questions, we can observe small correlations between readability and attention-related factors. Specifically, there is correlation with *alerting* (ρ = 0.234) and *orienting* (ρ = 0.208). In addition, there are negative correlations with *immediate recall* (ρ = -0.189) and *programming experience* (ρ = -0.073). When the p-values are adjusted with the Benjamini and Hochberg procedure (Benjamini and Hochberg, 1995), however, we observe no significant correlation.

We report the generalized linear model obtained for *readability* in Table 4 (right part). In this case, *programming experience* is not an important factor because the p-value is not significant (0.655). In the full model, we can see that the only significant factor is *orienting* (p-value = 0.048). This factor is important also in the minimal model (p-value = 0.012).

At the end, we report in Table 5 the results of the comparisons between distributions of independent variables in the groups *readable* and *unreadable* solutions. In terms of effect size, we have several variables for which the difference is non-negligible: *working memory* and *programming experience* (*medium*), *orienting* and *working memory* (*large*). The adjusted p-values, however, show no significant difference. This is most likely due to the lower number of data points considered with respect to the two other dependent variable, given that we only considered feature implementation tasks for *readability*.

Summary of RQ$_3$. *Orienting* is significantly associated with the code readability in the generalized linear model, and orienting of developers who produce readable code is largely higher compared to the ones who produce unreadable code. However, such a difference is most likely not significant because of the lower number of data-points considered for this RQ.

5.4 Discussion

In our study, we aimed at assessing to what extent cognitive aspects are correlated with developers' performance. In particular, we investigated to what extent attention and memory correlate with the time required to complete a development task and the quality of the solution, *i.e.*, both in terms of correctness and readability. The empirical evidence provided by our study suggests a negative result for memory and attention. Indeed, neither attention nor memory seem to explain how developers complete coding tasks in terms of correctness. While we obtain a negative result for correctness, we obtain a positive result for time and readability. This result concurs with the previous studies (Peitek et al., 2018; Siegmund et al.,

Table 5: Comparisons between independent variables in groups (correct vs incorrect for RQ_1, time-up vs non-time-up for RQ_2, and readable vs unreadable for RQ_3) using Mann-Withney (†) and Fisher (‡) tests.

Comparison	**Variable**	***p*-value**	**Cliff's *d***
Correct *vs.* Incorrect	Alerting†	0.973	-0.039 (negl.)
	Orienting†	0.973	-0.038 (negl.)
	Executive control†	0.973	0.048 (negl.)
	Immediate recall†	0.973	0.077 (negl.)
	Working memory†	0.973	0.022 (negl.)
	Programming experience†	0.154	**0.265** (small)
	Task difficulty‡	1.000	0.020 (negl.)
	Task type‡	0.973	-0.102 (negl.)
Time-up *vs.* Non-time-up	Alerting†	0.746	-0.093 (negl.)
	Orienting†	0.381	**0.174** (small)
	Executive control†	0.932	0.009 (negl.)
	Immediate recall†	0.932	0.029 (negl.)
	Working memory†	0.920	**-0.059** (negl.)
	Programming experience†	**0.010**	**-0.323** (small)
	Task difficulty‡	0.746	0.097 (negl.)
	Task type‡	0.932	0.032 (negl.)
Readable *vs.* Unreadable	Alerting†	1.000	0.089 (negl.)
	Orienting†	0.447	**0.782** (large)
	Executive control†	1.000	-0.056 (negl.)
	Immediate recall†	1.000	-0.064 (negl.)
	Working memory†	0.912	**-0.379** (medi.)
	Programming experience†	0.912	**-0.355** (medi.)
	Task difficulty‡	1.000	0.016 (negl.)

2017) that showed that the attention is an important factor for program comprehension. Given these positive results and the importance of code readability for program comprehension, we can conclude that developers with high attention not only understand code better, but also write more readable code. We did not observe a correlation between readability and memory-related factors. It is worth noting that explaining the outcome of the activity of code writing, either for implementing new features or for fixing bugs, is an ambitious task, as the low explanatory power obtained through our models for *correctness* show. This means that, probably, many other factors (including other cognitive ones) should be taken into account to achieve results usable in practice. In addition, the complexity of relation between cognitive factors and outcome of coding task and sample characteristics are an important limitation of the measurement of attention is that these results could not be generalized on all personal characteristics. Despite this limitation, we include other related cognitive constructs, *i.e.*, working memory, to guarantee inhibitory control constructs (Colom et al., 2005; Martínez et al., 2011). As for the measurement of working memory, a possible limitation is that we do not measure the state of mind of developers (*e.g.*, depression and fatique) (Genova et al., 2009).

As for correctness and time, our results show that there is a single variable that clearly outperforms all the others: the *programming experience* in the specific programming language (Java, in our study). Therefore, we can say, from our results, that, while there are some weak relations between cognitive aspects and the one variable we considered, *i.e.*, *correctness* and *time*, experience alone is a far better variable to explain both of them. In other words, there appears to be no "shortcut" for improving in coding tasks from the point of view of cognitive human aspects. However, the association observed between *readability* and *attention* suggests that it should be possible to devise a training that allow developers to improve cognitive functions (Oded, 2011) (attention, in our case). Such an improvement could be beneficial for writing readable code. However, more research is needed in this direction to prove that this is a concrete possibility. We tried to run the same analysis on a subset of our sample composed only of Bachelor students to understand if there is any difference with a broader population composed also of professional developers. We observed that most of the results are in line with the ones obtained on the whole sample. When we consider readability as the outcome variable, however, we observe that "Orienting" is only slightly significant (p=0.056 instead of 0.012 obtained on the whole sample). This is likely due to the fewer data points considered.

Another interesting phenomenon we observed is the following: In general, individual differences (in this case, in terms of programming experience) allow to better explain the outcome than the differences among the tasks (in our study, type and difficulty).

This means that the skills of a developer are far more important than the characteristics of the task at hand for determining the success in completing it and the time required. Also, if we analyze only two type of tasks, this is in line with what was previously observed for code understandability (Scalabrino et al., 2019). We can infer that developers with low experience that work on a task might take more time and might produce worse results *regardless of the task at hand*. This allows us to give a clear recommendation to practitioners, which is already applied in many contexts because of anecdotal evidence: *Less experienced developers should benefit from more experienced developers to achieve higher quality in the final software product.*

6 Threats to Validity

As it is the case for any empirical study validity of our conclusions might have been threatened in several ways.

6.1 Threat to Construct Validity

The biggest threats to construct validity are related to the methodology used to measure the independent variables, above all the attention- and memory-related ones. As described in Section 4, we use state-of-the-art psychometric tests to achieve this goal and we replicated them in our web-app. It might be possible that implementation errors have caused wrong measurements. Recent work Brearly et al. (2018) has highlighted that different platforms can also produce different

tests results. Thus, the problems related to the use of a self-implemented web-app are more complex than the correctness of the implementation. However, as we report later, the cognitive variables we measured on our sample are in line with the ones from other populations. This increases our confidence in the correct implementation of the platform we developed. We thoroughly tested the web-app to avoid this. In addition, there is the possibility that we use existing psychometric tests in a wrong way. Attention and memory could be measured with dedicated and specialized psychometric tests who are far from our knowledge. The accuracy with which we could reliably measure reaction times in ANT (for attention-related metrics) was within tenths of a second; state-of-the-art measurement provide the measure of reaction times with the precision of thousandths of a second. We believe that this is a negligible limitation: the mean reaction time we obtained is $\sim 681ms$, meaning that the maximum error due to the lack of precision ($\pm$50ms) would be at most $\sim$ 7.3%. Such a level of precision would be mostly irrelevant. The original version of ANT (Fan et al., 2002) we used for measuring attention-related factors provided three blocks with 96 evaluations each (288 total evaluations). We used reduced version of such a test (Weaver et al., 2013), which provide two blocks of 64 evaluations each (128 total evaluations). This allowed us to reduce the effort for developers. It could be possible that psychometric tests might cause fatigue in the developer and, therefore, reduce their ability of correctly completing the tasks or increasing the time required to do so. To reduce this effect, we let developer rest for a few minutes after the tests, before starting the tasks. We could not avoid this, since measurements of attention-related factor may depend on the context in which they are taken (Matchock and Mordkoff, 2009).

Another possible problem could be related to the operationalization and measurement of attention and memory. Both such aspects are acquired through psychometric tests, which could not be administered while the developer was writing code to avoid disctraction (thus an impact on the outcome). This limitation, however, could impact the reliability of measurements since developers' attention and memory could drop during the task. To reduce the impact of such a limitation, we measured these aspects before each session, as previous studies do (Oliveira et al., 2018; Brun et al., 2021).

Another possible threat is related to the representativeness of the sample because of the convenience sampling we used. We compared the distribution of the cognitive qualities from our sample with the cognitive distribution from a previous study (Lachman et al., 2014). We found that our results are in line with cognitive distribution from such a study. As for the immediate recall, we compare the distribution obtained on our sample (mean and standard deviation) with the one obtained by Lachman et al. (2014). In our study, the mean immediate recall is 6.26 (sd=2.37), while in the population studied by Lachman *et al.* it was 6.74 (sd=2.28), despite the number of participants is much larger, *i.e.*, 7,100 vs our 32 (two measurements for each participant, thus 64). As for the working memory, we compared our distribution with the one obtained by Fellows and Schmitter-Edgecombe (2020). In our population, the mean working memory is 82.5 (sd=23.8), in the study by Fellows *et al.*, who involved 536 participants, it was 61.53 (sd=14.08). Despite the difference, in terms of mean, it can be observed that both the measures are comparable given the quite large standard deviation, particularly for our sample, which is smaller. As for the attention level, we compare the results with the ones obtained by Weaver et al. (2013). In this case, the mean

is 660.4 (sd=121.4) for the "Java-ANT" group, 662.5 (sd=96.2) for the "CRSD-ANT" group, while in our study it is 662.06 (sd=202.58). Also in this case, the results are in line with the ones obtained in the literature.

6.2 Threat to Internal Validity

A threats to internal validity is related to the measurement of *programming experience*. We asked developers to indicate the number of years of experience in Java. Siegmund et al. (2014) showed that, while this is the most commonly used approach, other self-estimation questions might be more relevant for students (*e.g.*, self-assessment of experience compared to other class mates). Therefore, alternative and more reliable measures of *programming experience* might allow to achieve even higher correlations with both *correctness* and *time*.

Another threat is related to the definition of the tasks. It could be argued that the bug-injection methodology we used is not completely realistic since we arbitrarily modified the code. However, a similar methodology has been adopted in previous studies as well (Hutchins et al., 1994; Wong et al., 1998, 2013). Also, it could be argued that the description of the problems taken from LeetCode was not necessarily understandable to all developers. We assume that, being exposed every day to the many users of LeetCode, any understandability problem was removed from the description in time.

Another threat is related to the task choice. We chose tasks considering different difficulty levels: However, the perceived difficulty is subjective by nature. For example, the task *Find and Replace Pattern* could be easy for a developer that has experience with regular expressions, but it could be difficult for others.

For psychometric tests there could be other cognitive human factors to be considered. For example, we could consider the measurements of natural language competencies to explain code readability.

There could be other factors, different from attention and associated with natural language competencies, which better explain code readability. For instance, proficiency in English writing, knowledge of the English vocabulary. Hence, the recommendation to devise exercises to improve attention in order to have a positive effect on code readability might be overridden by the need to improve English language knowledge.

The problem of lack of control on the programming experience could be also related to the self-selected convenience sampling and one possible interference of demographics. This is related to the fact that our sample is very homogeneous and their demographics have limited variance. As described in Section 4, we recruited 32 participants without considering their programming experience. As written above, we invited participants from four different countries. As a consequence, their background is heterogeneous. Unfortunately, given the lack of control, there is a developer with 15 years of Java experience that increase the average experience. However, if we removed the outlier, we do not obtain dissimilar results.

Another threat is induced by the time-out we used for the tasks (30 minutes). Such a time could not have been necessary to complete the entire software development task for some developers. We kept a time limit of 30 minutes because participants to the pilot study confirmed that the time was sufficient to complete the tasks at hand. For this reason, and also to decrease fatigue effects, we decided

to adopt this time. We verified if there are possible effects of fatigue and learning. To do this, we analyzed results of Day 1 and results of Day 2. For *correctness* and *time*, we observed a slight difference in the two sessions: On Day 1, 69% of solutions contain bugs and 59% of solutions have been completed in the given time. On Day 2, 79% of solutions contain bugs and 67% of solutions have been completed in the given time. As for *readability*, we observed no relevant difference: 45% of the solutions were readable for Day 1 and 46% of solution were readable for Day 2. Despite such a difference, given our balanced design, we believe that these effect did not bias the final results.

Another limitation is related to the choice of the programming language. There is the possibility that we chose a unfamiliar language for some developers. We used Java since all the participants attended a University course on such a programming language.

6.3 Threat to External Validity

Our findings may be mostly related to the specific sample of developers we involved in the study. We included a total of 32 developers from four different countries. In this way, we believe we limited the biases related to common education and social background. Our a-priori power analysis suggested that a sample of 26 developers would have been sufficient to achieve 80% power. To further verify that our sample is not too small, we also run post-hoc power analysis for our two regression models. We achieve 83.1% power for the *correctness* model (RQ_1), 99.9% for the *time* model (RQ_2) and 95.0% for the *readability* model (RQ_3). Therefore, we can confidently conclude that the data on which we based our analyses are sufficient to draw the conclusions we made.

The only hints at the fact a bigger sample could have resulted in additional significant differences are given by (i) the non-negligible effect size we obtained when comparing the *orienting* values between the *time-up* and the *non-time-up* groups, and (ii) the almost significant p-value of *alerting* obtained in the full regression model that predicts *time* (p-value = 0.056). In the former, the p-value of the comparison (0.38) is quite far from the threshold we chose for significance (0.05), and the effect size, while being *small* (0.174), is very close to the *negligible* threshold (0.148). As for the latter, instead, it can be noticed, given the coefficient assigned to the *alerting*, such a value has an influence on the time fitted by the model between -3.6 and +1.2 minutes. For comparison, the effect of *programming experience* is between -18.8 and -1.3 minutes. In other words, it would require developers to have a relatively high *alerting* efficiency for having a small benefit in terms of time, according to the model, regardless of the significance of the variable.

Another important point is how much the selected tasks are valid representations for real coding problems. Selected tasks do not represent all the possible coding tasks, but only a part of low-level coding tasks (*e.g.*, implementation of method). For example, we ask the participants to implement a regular expression matching algorithm, but we do not ask them to write a regular expression to filter out people whose name does not meet some criteria. However, the tasks we selected represent realistic coding problems that thevelopers might find themselves solving. For example, in the "Duplicate zeros" task, we ask them to "duplicate each occurrence of zero, shifting the remaining elements to the right." There are

plenty of contexts in which developers might need to insert elements in an array and shifting the remaining ones to the right[4].

7 Conclusion

Among cognitive aspects, attention and memory have been shown to be related to the outcome of different kinds of tasks (*e.g.*, driving or solving mathematical problems), and they have also been used in the context of Software Engineering and Software Security (*e.g.*, for determining the usability of security APIs, Oliveira et al. (2018)), but they are also used to observe neural activation related to *attention* and *working memory* during coding activities (Peitek et al., 2018; Siegmund et al., 2014; Krueger et al., 2020).

For this reason, we theorize that attention- and memory-related factors also play a role in coding tasks, which are at the base of software development and evolution. We conducted a controlled experiment with 32 developers, and asked them to complete 4 tasks each. We measured three attention-related metrics and two memory-related ones, widely used in the literature. We aim at predicting the *time* needed to complete the tasks and the quality of the solutions, both in terms of *correctness* and *readability*. We check the relationship between independent variables (including also *programming experience*, *task difficulty*, and *task type*) and the three dependent ones both by using single correlations and by combining them in a regression model. On the one hand, we obtained a *negative result*: Neither attention- nor memory-related factors play a role in explaining *correctness*. Programming experience is the only significant factor, much more important than the task difficulty and the task type. On the other hand, we observed that attention-related factors (alerting and orienting) are associated with higher capability of writing readable solutions and of completing solutions in time.

Our results clearly show that attention and memory play a marginal role in explaining *correctness*, if any. Replications of our study are needed to further corroborate our findings. Specifically, broader studies are needed to show possible small interaction that we could not get at this scale. Also, future research could focus on other cognitive aspects we did not consider in this study, such as *intelligence* (*e.g.*, through IQ).

Conflict of interest

The authors declare that they have no conflict of interest.

References

Allan J (2013) Cognitions, Springer New York, New York, NY, pp 441–441. DOI 10.1007/978-1-4419-1005-9_1114, URL `https://doi.org/10.1007/978-1-4419-1005-9_1114`

[4] See, for example: `https://github.com/FantacyWorld/libMSVL/blob/master/dyn_array.c\#L307`.

Baddeley AD (1983) Working memory. Philosophical Transactions of the Royal Society of London B, Biological Sciences 302(1110):311–324

Benedict RH, DeLuca J, Phillips G, LaRocca N, Hudson LD, Rudick R, Consortium MSOA (2017) Validity of the symbol digit modalities test as a cognition performance outcome measure for multiple sclerosis. Multiple Sclerosis Journal 23(5):721–733

Benjafield JG, Smilek D, Kingstone A (2010) Cognition (4th ed.). New York: Oxford University Press

Benjamini Y, Hochberg Y (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing. Journal of the Royal statistical society: series B (Methodological) 57(1):289–300

Bhattacharya P, Neamtiu I (2011) Bug-fix time prediction models: can we do better? In: Proceedings of the 8th Working Conference on Mining Software Repositories, pp 207–210

Bialystok E, DePape AM (2009) Musical expertise, bilingualism, and executive functioning. Journal of Experimental Psychology: Human Perception and Performance 35(2):565

Blackburn JD, Scudder GD, Van Wassenhove LN (1996) Improving speed and productivity of software development: a global survey of software developers. IEEE transactions on software engineering 22(12):875–885

Blank A, Frush Holt R, Pisoni DB, Kronenberger WG (2020) Associations between parenting stress, language comprehension, and inhibitory control in children with hearing loss. Journal of Speech, Language, and Hearing Research 63(1):321–333

Bowrin AR, King J (2010) Time pressure, task complexity, and audit effectiveness. Managerial auditing journal

Brearly TW, Rowland JA, Martindale SL, Shura RD, Curry D, Taber KH (2018) Comparability of ipad and web-based nih toolbox cognitive battery administration in veterans. Archives of Clinical Neuropsychology 34(4):524–530

Brun Y, Lin T, Somerville JE, Myers E, Ebner NC (2021) Blindspots in python and java apis result in vulnerable code. arXiv preprint arXiv:210306091

Cairncross M, Gindwani H, Rita Egbert A, Torres IJ, Hutchison JS, Dams O'Connor K, Panenka WJ, Brubacher JR, Meddings L, Kwan L, et al. (2022) Criterion validity of the brief test of adult cognition by telephone (btact) for mild traumatic brain injury. Brain Injury pp 1–9

Cappos J, Zhuang Y, Oliveira D, Rosenthal M, Yeh KC (2014) Vulnerabilities as blind spots in developer's heuristic-based decision-making processes. In: Proceedings of the 2014 New Security Paradigms Workshop, pp 53–62

Cliff N (1993) Dominance statistics: Ordinal analyses to answer ordinal questions. Psychological bulletin 114(3):494

Colom R, Flores-Mendoza C, Quiroga MÁ, Privado J (2005) Working memory and general intelligence: The role of short-term storage. Personality and Individual Differences 39(5):1005–1014

Conway A, Jarrold C, Miyake A (2008) Variation in working memory. Oxford University Press

Coulter NS (1983) Software science and cognitive psychology. IEEE Transactions on Software Engineering (2):166–171

Crawford JR (1998) Introduction to the assessment of attention and executive functioning. Neuropsychological rehabilitation 8(3):209–211

Di Nucci D, Palomba F, De Rosa G, Bavota G, Oliveto R, De Lucia A (2017) A developer centered bug prediction model. IEEE Transactions on Software Engineering 44(1):5–24

Ebisch SJ, Mantini D, Romanelli R, Tommasi M, Perrucci MG, Romani GL, Colom R, Saggino A (2013) Long-range functional interactions of anterior insula and medial frontal cortex are differently modulated by visuospatial and inductive reasoning tasks. Neuroimage 78:426–438

Efroymson MA (1960) Multiple regression analysis. Mathematical methods for digital computers pp 191–203

Eriksen BA, Eriksen CW (1974) Effects of noise letters upon the identification of a target letter in a nonsearch task. Perception & psychophysics 16(1):143–149

Fan J, McCandliss BD, Sommer T, Raz A, Posner MI (2002) Testing the efficiency and independence of attentional networks. Journal of cognitive neuroscience 14(3):340–347

Fellows RP, Schmitter-Edgecombe M (2020) Symbol digit modalities test: Regression-based normative data and clinical utility. Archives of Clinical Neuropsychology 35(1):105–115

Fisher RA (1922) On the interpretation of χ 2 from contingency tables, and the calculation of p. Journal of the Royal Statistical Society 85(1):87–94

Floyd B, Santander T, Weimer W (2017) Decoding the representation of code in the brain: An fmri study of code review and expertise. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE, pp 175–186

Forn C, Belloch V, Bustamante JC, Garbin G, Parcet-Ibars MÀ, Sanjuan A, Ventura N, Ávila C (2009) A symbol digit modalities test version suitable for functional mri studies. Neuroscience letters 456(1):11–14

Forn C, Rocca MA, Boscá I, Casanova B, Sanjuan A, Filippi M (2013) Analysis of "task-positive" and "task-negative" functional networks during the performance of the symbol digit modalities test in patients at presentation with clinically isolated syndrome suggestive of multiple sclerosis. Experimental brain research 225(3):399–407

Friedman NP, Miyake A (2004) The relations among inhibition and interference control functions: a latent-variable analysis. Journal of experimental psychology: General 133(1):101

Gellman MD, Turner JR (eds) (2013) Cognition, Springer New York, New York, NY, pp 441–441. DOI 10.1007/978-1-4419-1005-9_100314, URL `https://doi.org/10.1007/978-1-4419-1005-9_100314`

Genova HM, Hillary FG, Wylie G, Rypma B, Deluca J (2009) Examination of processing speed deficits in multiple sclerosis using functional magnetic resonance imaging. Journal of the International Neuropsychological Society 15(3):383–393

Gershon RC, Wagster MV, Hendrie HC, Fox NA, Cook KF, Nowinski CJ (2013) Nih toolbox for assessment of neurological and behavioral function. Neurology 80(11 Supplement 3):S2–S6

Green CS, Bavelier D (2007) Action-video-game experience alters the spatial resolution of vision. Psychological science 18(1):88–94

Habib M, Besson M (2009) What do music training and musical experience teach us about brain plasticity? Music Perception 26(3):279–285

Halberda J, Mazzocco MM, Feigenson L (2008) Individual differences in non-verbal number acuity correlate with maths achievement. Nature 455(7213):665–668

He Z, Shu F, Yang Y, Li M, Wang Q (2012) An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering 19(2):167–199

Howard SJ, Johnson J, Pascual-Leone J (2014) Clarifying inhibitory control: Diversity and development of attentional inhibition. Cognitive Development 31:1–21

Huang Y, Liu X, Krueger R, Santander T, Hu X, Leach K, Weimer W (2019) Distilling neural representations of data structure manipulation using fmri and fnirs. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 396–407

Hughes DJ (2018) Psychometric validity: Establishing the accuracy and appropriateness of psychometric measures. The Wiley handbook of psychometric testing: A multidisciplinary reference on survey, scale and test development pp 751–779

Hutchins M, Foster H, Goradia T, Ostrand T (1994) Experiments on the effectiveness of dataflow-and control-flow-based test adequacy criteria. In: Proceedings of 16th International conference on Software engineering, IEEE, pp 191–200

Juristo N, Moreno AM (2013) Basics of software engineering experimentation. Springer Science & Business Media

Karas Z, Jahn A, Weimer W, Huang Y (2021) Connecting the dots: rethinking the relationship between code and prose writing with functional connectivity. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp 767–779

Koen JD, Aly M, Wang WC, Yonelinas AP (2013) Examining the causes of memory strength variability: Recollection, attention failure, or encoding variability? Journal of Experimental Psychology: Learning, Memory, and Cognition 39(6):1726

Krueger R, Huang Y, Liu X, Santander T, Weimer W, Leach K (2020) Neurological divide: an fmri study of prose and code writing. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), IEEE, pp 678–690

Kuutila M, Mäntylä MV, Claes M, Elovainio M (2017) Reviewing literature on time pressure in software engineering and related professions: computer assisted interdisciplinary literature review. In: 2017 IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion), IEEE, pp 54–59

Lachman ME, Agrigoroaei S, Tun PA, Weaver SL (2014) Monitoring cognitive functioning: psychometric properties of the brief test of adult cognition by telephone. Assessment 21(4):404–417

LeetCode (2020a) Leetcode problem: Duplicate zeros. URL `https://leetcode.com/problems/duplicate-zeros/`

LeetCode (2020b) Leetcode problem: Find and replace pattern. URL `https://leetcode.com/problems/find-and-replace-pattern/`

LeetCode (2020c) Leetcode problem: Regular expression matching. URL `https://leetcode.com/problems/regular-expression-matching/`

LeetCode (2020d) Leetcode problem: Sort the matrix diagonally. URL `https://leetcode.com/problems/sort-the-matrix-diagonally/`

Lesage E, Sutherland MT, Ross TJ, Salmeron BJ, Stein EA (2020) Nicotine dependence (trait) and acute nicotinic stimulation (state) modulate attention but not inhibitory control: converging fmri evidence from go–nogo and flanker tasks. Neuropsychopharmacology 45(5):857–865

Li Z, Jing XY, Zhu X (2018) Progress on approaches to software defect prediction. Iet Software 12(3):161–175

Lilienthal L, Tamez E, Shelton JT, Myerson J, Hale S (2013) Dual n-back training increases the capacity of the focus of attention. Psychonomic bulletin & review 20(1):135–141

Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. The annals of mathematical statistics pp 50–60

Martínez K, Burgaleta M, Román FJ, Escorial S, Shih PC, Quiroga MÁ, Colom R (2011) Can fluid intelligence be reduced to 'simple'short-term storage? Intelligence 39(6):473–480

Matchock RL, Mordkoff JT (2009) Chronotype and time-of-day influences on the alerting, orienting, and executive components of attention. Experimental brain research 192(2):189–198

Mundy E, Gilmore CK (2009) Children's mapping between symbolic and nonsymbolic representations of number. Journal of experimental child psychology 103(4):490–502

Musso M, Kyndt E, Cascallar E, Dochy F (2012) Predicting mathematical performance: The effect of cognitive processes and self-regulation factors. Education Research International 2012

Nour S, Struys E, Stengers H (2019) Attention network in interpreters: The role of training and experience. Behavioral Sciences 9(4):43

Oded Y (2011) Biofeedback-based mental training in the military—the "mental gym™" project. Biofeedback 39(3):112–118

Oliveira D, Rosenthal M, Morin N, Yeh KC, Cappos J, Zhuang Y (2014) It's the psychology stupid: how heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots. In: Proceedings of the 30th Annual Computer Security Applications Conference, pp 296–305

Oliveira DS, Lin T, Rahman MS, Akefirad R, Ellis D, Perez E, Bobhate R, DeLong LA, Cappos J, Brun Y (2018) API blindspots: Why experienced developers write vulnerable code. In: Fourteenth Symposium on Usable Privacy and Security ({SOUPS} 2018), pp 315–328

Parmenter B, Weinstock-Guttman B, Garg N, Munschauer F, Benedict RH (2007) Screening for cognitive impairment in multiple sclerosis using the symbol digit modalities test. Multiple Sclerosis Journal 13(1):52–57

Passolunghi MC, Vercelloni B, Schadee H (2007) The precursors of mathematics learning: Working memory, phonological ability and numerical competence. Cognitive development 22(2):165–184

Peitek N, Siegmund J, Apel S, Kästner C, Parnin C, Bethmann A, Leich T, Saake G, Brechmann A (2018) A look into programmers' heads. IEEE Transactions on Software Engineering 46(4):442–462

Piantadosi V, Scalabrino S, Serebrenik A, Novielli N, Oliveto R (2021) Replication package of "do attention and memory explain the performance of software developers?". URL `https://figshare.com/s/f3cf009d98ac60530ec6`

Pinto G, Rebouças M, Castor F (2017) Inadequate testing, time pressure, and (over) confidence: a tale of continuous integration users. In: 2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), IEEE, pp 74–77

Posner MI (1980) Orienting of attention. Quarterly journal of experimental psychology 32(1):3–25

Posner MI, Petersen SE (1990) The attention system of the human brain. Annual review of neuroscience 13(1):25–42

Posnett D, D'Souza R, Devanbu P, Filkov V (2013) Dual ecological measures of focus in software development. In: 2013 35th International Conference on Software Engineering (ICSE), IEEE, pp 452–461

Rahman F, Posnett D, Devanbu P (2012) Recalling the" imprecision" of cross-project defect prediction. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, pp 1–11

Rapport MD, Kofler MJ, Alderson RM, Timko Jr TM, DuPaul GJ (2009) Variability of attention processes in adhd: Observations from the classroom. Journal of Attention Disorders 12(6):563–573

Rasch RH, Tosi HL (1992) Factors affecting software developers' performance: An integrated approach. MIS quarterly pp 395–413

Ricca F, Di Penta M, Torchiano M, Tonella P, Ceccato M (2007) The role of experience and ability in comprehension tasks supported by uml stereotypes. In: 29th International Conference on Software Engineering (ICSE'07), IEEE, pp 375–384

Ricca F, Di Penta M, Torchiano M, Tonella P, Ceccato M (2009) How developers' experience and ability influence web application comprehension tasks supported by uml stereotypes: A series of four experiments. IEEE Transactions on Software Engineering 36(1):96–118

Roy E (2013) Cognitive Function, Springer New York, New York, NY, pp 448–449. DOI 10.1007/978-1-4419-1005-9_1117, URL `https://doi.org/10.1007/978-1-4419-1005-9_1117`

Scalabrino S, Linares-Vásquez M, Oliveto R, Poshyvanyk D (2018) A comprehensive model for code readability. Journal of Software: Evolution and Process 30(6):e1958

Scalabrino S, Bavota G, Vendome C, Poshyvanyk D, Oliveto R, et al. (2019) Automatically assessing code understandability. IEEE Transactions on Software Engineering

Schacter DL, Wagner AD, Buckner RL (2000) Memory systems of 1999.

Schellenberg EG (2004) Music lessons enhance iq. Psychological science 15(8):511–514

Sharafi Z, Huang Y, Leach K, Weimer W (2021) Toward an objective measure of developers' cognitive activities. ACM Transactions on Software Engineering and Methodology (TOSEM) 30(3):1–40

Shneiderman B, Mayer R (1979) Syntactic/semantic interactions in programmer behavior: A model and experimental results. International Journal of Computer & Information Sciences 8(3):219–238

Shorey C, Friedman E (2018) Multimorbidity and cognitive decline in a national sample of aging adults. Innovation in Aging 2(Suppl 1):505

Siegmund J, Kästner C, Liebig J, Apel S, Hanenberg S (2014) Measuring and modeling programming experience. Empirical Software Engineering 19(5):1299–1334

Siegmund J, Peitek N, Parnin C, Apel S, Hofmeister J, Kästner C, Begel A, Bethmann A, Brechmann A (2017) Measuring neural efficiency of program comprehension. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp 140–150

Silva P, Spedo C, Barreira AA, Leoni RF (2018) Symbol digit modalities test adaptation for magnetic resonance imaging environment: A systematic review and meta-analysis. Multiple sclerosis and related disorders 20:136–143

Sloan L (2015) Learn about Spearman's Rank-order Correlation Coefficient in SPSS with Data from the General Social Survey (2012). SAGE Publications

Song MK, Ward SE, Bair E, Weiner LJ, Bridgman JC, Hladik GA, Gilet CA (2015) Patient-reported cognitive functioning and daily functioning in chronic dialysis patients. Hemodialysis International 19(1):90–99

Spearman C (1961) The proof and measurement of association between two things. American Journal of Psychology

Thota MK, Shajin FH, Rajesh P, et al. (2020) Survey on software defect prediction techniques. International Journal of Applied Science and Engineering 17(4):331–344

Tiego J, Testa R, Bellgrove MA, Pantelis C, Whittle S (2018) A hierarchical model of inhibitory control. Frontiers in psychology 9:1339

Tun PA, Lachman ME (2006) Telephone assessment of cognitive function in adulthood: the brief test of adult cognition by telephone. Age and Ageing 35(6):629–632

Wang H, Fan J, Yang Y (2004) Toward a multilevel analysis of human attentional networks. In: Proceedings of the Annual Meeting of the Cognitive Science Society, vol 26

Weaver B, Bedard M, McAuliffe J, Parkkari M (2009) Using the attention network test to predict driving test scores. Accident Analysis & Prevention 41(1):76–83

Weaver B, Bédard M, McAuliffe J (2013) Evaluation of a 10-minute version of the attention network test. The Clinical Neuropsychologist 27(8):1281–1299

Wei W, Yuan H, Chen C, Zhou X (2012) Cognitive correlates of performance in advanced mathematics. British Journal of Educational Psychology 82(1):157–181

Wong WE, Horgan JR, London S, Mathur AP (1998) Effect of test set minimization on fault detection effectiveness. Software: Practice and Experience 28(4):347–369

Wong WE, Debroy V, Gao R, Li Y (2013) The dstar method for effective software fault localization. IEEE Transactions on Reliability 63(1):290–308

Woumans E, Ceuleers E, Van der Linden L, Szmalec A, Duyck W (2015) Verbal and nonverbal cognitive control in bilinguals and interpreters. Journal of Experimental Psychology: Learning, Memory, and Cognition 41(5):1579

Zelazo PD, Anderson JE, Richler J, Wallner-Allen K, Beaumont JL, Weintraub S (2013) Ii. nih toolbox cognition battery (cb): Measuring executive function and attention. Monographs of the Society for Research in Child Development 78(4):16–33

Zhang H, Gong L, Versteeg S (2013) Predicting bug-fixing time: an empirical study of commercial software projects. In: 2013 35th International Conference on Software Engineering (ICSE), IEEE, pp 1042–1051

Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp 91–100