Program Comprehension and Code Complexity Metrics: An fMRI Study

Norman Peitek
Leibniz Institute for Neurobiology
Magdeburg, Germany

Sven Apel
Saarland University, Saarland Informatics Campus
Saarbrücken, Germany

Chris Parnin
NC State University
Raleigh, North Carolina, USA

André Brechmann
Leibniz Institute for Neurobiology
Magdeburg, Germany

Janet Siegmund
Chemnitz University of Technology
Chemnitz, Germany

Abstract*—*Background: **Researchers and practitioners have been using code complexity metrics for decades to predict how developers comprehend a program. While it is plausible and tempting to use code metrics for this purpose, their validity is debated, since they rely on simple code properties and rarely consider particularities of human cognition.**
Aims: **We investigate whether and how code complexity metrics reflect difficulty of program comprehension.**
Method: **We have conducted a functional magnetic resonance imaging (fMRI) study with 19 participants observing program comprehension of short code snippets at varying complexity levels. We dissected four classes of code complexity metrics and their relationship to neuronal, behavioral, and subjective correlates of program comprehension, overall analyzing more than 41 metrics.**
Results: **While our data corroborate that complexity metrics can—to a limited degree—explain programmers' cognition in program comprehension, fMRI allowed us to gain insights into *why* some code properties are difficult to process. In particular, a code's textual size drives programmers' attention, and vocabulary size burdens programmers' working memory.**
Conclusion: **Our results provide neuro-scientific evidence supporting warnings of prior research questioning the validity of code complexity metrics and pin down factors relevant to program comprehension.**
Future Work: **We outline several follow-up experiments investigating fine-grained effects of code complexity and describe possible refinements to code complexity metrics.**

Prelude

Let us begin with a short subjective assessment of code complexity. Consider the following code snippet[1]:

```
1 public List<Integer> compute(int a, int b, int c) {
2     if (a > b) { int temp = b; b = a; a = temp; }
3     if (a > c) { int temp = c; c = a; a = temp; }
4     if (b > c) { int temp = c; c = b; b = temp; }
5     return Arrays.asList(a, b, c);
6 }
```

Before going further—putting aside concerns of performance or coding style—we would like you to spend a few moments and calculate the result of `compute(8,12,4)`. Based on this experience, please rate the *complexity* (i.e., how difficult it is to understand the parts of the code and their dependencies) of this code snippet on a 5-point scale, from *Very Simple* to *Very Complex*.

Well, how did it go? How did you rate this snippet? Software engineering research has developed several complexity metrics calculated based on various structural properties of the code [2]. For example, *Halstead's complexity*, a metric for vocabulary size [3], would yield a score of 21. McCabe's *cyclomatic complexity*, a metric for the number of linearly independent paths through a program [4], would yield a score of 4. Beyer's DepDegree [5], a metric that considers complexity arising from low-level data flow, results in a score of 31. Finally, a simple metric of counting lines of code [6] would provide a score of 7. Apparently, these values differ considerably, although it is difficult to tell without a common scale.

[1]The code snippet is an example of an unrolled insertion sort, which is fast for sorting a small number of inputs [1].

I. Introduction

In the past 40 years, the software engineering community has been using various complexity metrics to predict how programmers understand code [7], [8], [9], implement quality gates in continuous integration [10], [11], and predict the likelihood of defects [12], [13], [14]. For example, from 2010 to 2015, a total of 226 studies proposed or analyzed nearly 300 code metrics alone, with code complexity being one of the most frequent categories of study [2]. Many of these metrics are widely used in software analysis tools. For example, SonarQube, a popular static analysis tool used in continuous integration, supports a multitude of metrics, such as cyclomatic complexity, across dozens of programming languages. For this particular metric, the tool will report a "Methods should not be too complex" violation for any method that exceeds the default threshold of 10.

While code metrics help to describe properties of code, they are notoriously limited in capturing human cognition and behavior: Already 15 years ago, Kaner and Bond warned that too simplistic software metrics could do more harm than good because it is doubtful whether they actually measure what we think they measure [15]. Several studies underline this point. For example, Scalabrino et al. found in an empirical study, at most, minuscule correlations between complexity metrics and code understanding [16]. Ajami et al. found that complexity

metrics fail to consider how humans process code, for example, flat structures are easier to comprehend than nested ones [17]. In the same vein, Jbara and Feitelson provide evidence that complexity metrics miss the increased ease of comprehension of repeated code constructs [18]. Despite these warnings, it is tempting to use complexity metrics to predict how programmers perceive the complexity of code or how much cognitive load it would require to understand a piece of code.

Despite substantial research, the big picture is still unclear. So, the *overarching question* that drives our work is:

> Can variations in (classes of) code complexity metrics explain differences in programmer cognition during program comprehension?

To address this question, we draw on insights from neuroscience studies on cognitive load and from linguistic studies on sentence complexity. An extensive body of previous research has examined how syntax complexity in natural language can influence brain activation and processing difficulty. For example, complex sentences such as "The girl that the boy is tickling is happy" can be difficult to process for patients with Broca's aphasia [19] (a cognitive deficiency due to an acute brain lesion); these patients cannot reliably distinguish between the girl or the boy being happy. In healthy patients, such sentences will cause distinct activation of specialized language processing regions in the brain not seen with simpler grammatical structures. To perform these studies, neuro-linguists typically conduct a parameterized analysis of sentences, where they intentionally construct sentences that vary along several metrics of interest (such as left-branching complexity [20], movement distance [21], or filler-gap dependencies [22]) observing differences in behavioral measures and brain activation. Such studies provide deep insights into *why*—not only whether—comprehension of certain language constructs can be difficult, gaining insights into the inner processing of language cognition.

In the current work, we have conducted a study in which 19 participants comprehended 16 code snippets, which were systematically selected to contrast different values of code complexity metrics among four major classes of code complexity metrics: *code size*, *vocabulary size*, *control-flow complexity*, and *data-flow complexity*. For each, we picked a commonly used representative metric as a baseline before exploring 37 further metrics. The underlying idea is that, the more code lines (code size, LOC as representative) or vocabulary (vocabulary size, Halstead) to understand, or the more possible execution paths (control flow, McCabe) or data dependencies to keep track of (data flow, DepDegree), the more cognitive resources a programmer needs. To this end, *we explore the relation of code complexity metrics to behavioral and cognitive correlates of program comprehension*. We investigate the reliability of subjective perception of code complexity to include the programmer's perspective.

Based on their aim and definition, as well as prior neuro-linguistic studies, we expect different outcomes for different kinds of complexity metrics:

(a) a higher number of symbols (as measured by vocabulary-size metrics) induces higher cognitive processing demands, as seen when increasing the number of words in a sentence [23]. This is also supported by early studies on the relationship of Halstead's complexity and comprehensibility [7];
(b) an increased number of control paths (as measured by control-flow metrics) increases activation of areas associated with rule-guided conditional reasoning, such as reading conditional propositions [24] and counterfactual [25] sentences: "If Mike pressed the brake pedal, then the car would have stopped";
(c) a higher number of data-flow dependencies (as measured by DepDegree) increases activation of Broca's area (syntactic working memory) [22], as seen with sentences where an assignment of values must occur later in the sentence (i.e., filler-gap dependencies), "Which cowgirl did Mary expect to have injured herself due to negligence?".

In a nutshell, we found that changes in complexity metrics can explain differences in programmer cognition to different extents. Size-based and vocabulary-based metrics correlate with the anticipation of work, whereas data-flow metrics correlate with higher cognitive demands in a network of brain areas activated during program comprehension. Simple control-flow metrics (e.g., number of branches) predict cognitive load better than more complex control-flow metrics (e.g., McCabe). In general, not many code complexity metrics incorporate data flow, which seem to capture a distinct aspect of cognition, showing a promising avenue of further work.

In summary, we make the following contributions:

- We devise a multi-modal experiment framework that links complexity metrics to cognitive processes of program comprehension.
- We provide empirical evidence for the relation of 41 code complexity metrics, on the one hand, and neuronal and behavioral correlates, on the other hand.
- We show that behavioral data are the best predictor for subjective complexity.
- We present a non-exact, refined replication of our previous fMRI study with a larger set of more varied and targeted code snippets [26].
- We share an online replication package[2] to share experiment design, tasks, and analysis protocols.[3]

II. Study Design

Our study of neuronal and behavioral correlates of code complexity metrics builds on an observation that we made in previous work: As a side product of studying another research question (identifying neuronal correlates of program comprehension), we found first indications of a measurable correlation between neuronal activity and some code complexity metrics [26]. However, this has not been explored further, so the existence and kind of relationship between complexity metrics and cognitive processes remained unclear. We specifically

[2]https://github.com/brains-on-code/fMRI-complexity-metrics-icse2021/
[3]Access to actual fMRI data is granted upon request.

designed our study to unveil this relationship, including a multi-modal experiment setup tailored to this question, with proper code snippets that vary across different metrics, and by analyzing a total of 41 metrics regarding their predictive power.

A. Research Goals

In this paper, we aim at answering the following research questions:

RQ$_1$: Do different (classes of) code complexity metrics correlate with programmers' behavior during program comprehension?

RQ$_2$: Do different (classes of) code complexity metrics correlate with programmers' cognitive load in terms of brain (de)activation during program comprehension?

RQ$_3$: Do different (classes of) code complexity metrics correlate with programmers' subjective perception of code complexity?

B. Pilot Studies

To answer our research questions, we carefully designed our experiment. First, we compiled a set of 50 Java code snippets by obtaining snippets from previous studies on program comprehension [27], [28], [29] and by augmenting this set by searching for code snippets with similar complexity. Second, from this pool of code snippets with a wide range of complexities, we selected the most suitable snippets for an fMRI study by running two pilot studies at the authors' institutions. We asked 7 pilot-study participants (2 professional programmers and 5 PhD students) to understand the snippets as quickly and accurate as possible and to verbally share their thoughts afterwards. Unlike in the fMRI scanner, we did not set a time limit per snippet, because the actual comprehension time is an important factor to select appropriate snippets for an fMRI study. In the pilot studies, we also asked for a subjective evaluation of each snippet's complexity, but, unlike in the subsequent fMRI study, we did not ask them to rank all presented snippets. Third, we selected snippets for the fMRI study that were associated with a range of complexity metrics values balancing our final selection across the four classes of complexity metrics.

For illustration, we show one of the snippets in Listing 1, which computes the length of the last word in a string: The code exhibits large values for some of the metrics, not being trivial to solve (pilot participants took about 45 seconds), while still staying within the 60 seconds limit allowed in the fMRI study. Feedback from participants indicated that they needed a high level of cognitive effort to understand the snippet, but could still succeed. In particular, they noted that the snippet did not allow them to take a "mental break". That is, it was difficult to analyze individual statements while keeping other statements in mind, and that they were unable to match the code to any known algorithm.

Finally, consistent with previous studies [27], [28], we excluded all context information from snippets to enforce bottom-up comprehension [30]. This way, we reduce the influence of previous experience on cognitive load, because employing a more efficient top-down approach [31] is impeded, and because expertise can moderate the relationship between complexity and performance [7]. In Table I, we provide information on the code snippets and code complexity metrics that we selected for the fMRI study (see the supplementary Web site for all code snippets and all 41 metrics).

In Table II, we show the correlations among the four metrics for all code snippets. Although we designed the snippets to vary in complexity, we were restricted by the requirements of the fMRI scanner (especially the limited screen size to show a maximum of 30 lines of code and that each snippet is comprehensible within 60 seconds). Thus, a certain correlation among the metrics is unavoidable.

```
1  public static void main(){
2    String text = "The quick brown fox jumps";
3    System.out.print(compute(text));
4  }
5
6  static int compute(String text){
7    int result = 0;
8    boolean flag = false;
9    for (int i = text.length() - 1; i >= 0; i--){
10     char c = text.charAt(i);
11     if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')){
12       flag = true;
13       result++;
14     } else {
15       if (flag)
16         break;
17       }
18   }
19
20   return result;
21 }
```

Listing 1: Complex code snippet that computes the length of the last word in a string. The output for this snippet is "5".

C. fMRI Study

1) Research Context: When brain regions activate or deactivate during program comprehension (or any other cognitive process), their oxygenation levels change. This is reflected in a change in the proportion of oxygenated and deoxygenated blood (blood-oxygenation-level-dependent (BOLD) response), both of which have different magnetic properties [32]. An fMRI scanner measures these changes and localizes them to specific brain areas [33], which are associated with different cognitive processes. Thus, we can associate activated brain areas to how participants proceeded in comprehending a snippet. In addition, research on the brain's default mode network (DMN) [34], [35] suggests that cognitive load is linked to the level of deactivation of the DMN during cognitively demanding tasks [36]. This way, we can determine the cognitive load of program comprehension during the tasks.

2) Participants: Participants were 19 (including one whose fMRI data had to be excluded from analysis due to excessive head movements) late undergraduate or graduate students at the University Magdeburg (2 female, 17 male, 26.47 $\pm$[4] 2.68 years old). We determined their programming experience based on a validated questionnaire [37]. We found that they had a medium

[4] $m \pm s$ denotes a mean of m and standard deviation of s.

TABLE I: Code snippets used in the study with four selected complexity metric scores and experimental results: behavioral data (correctness, time) and subjective complexity. A higher intensity of a cell's background color indicates a higher complexity. The histogram plots show the distribution of subjective complexity; skewness to the right represents higher subjective complexity.

	Snippet	Complexity Metrics				Experiment Results		
		Code Size (LOC)[1]	Vocabulary (Halstead)[1]	Control Flow (McCabe)[1]	Data Flow (DepDegree)[2]	Correctness (in %)	Time (in sec.)	Subjective Complexity Low → Medium → High
Loop	Average of array	17	12.64	3	17	47%	49.0	
	Contains substring	26	25.50	7	29	32%	50.2	
	Count vowels in string	19	13.00	5	22	89%	30.4	
	Greatest common divisor	24	26.63	5	33	47%	50.0	
	h index	21	16.25	4	20	53%	46.0	
	Length of last word	22	18.58	8	17	58%	43.3	
	Palindrome check	17	16.72	4	17	79%	38.4	
	Square root of array	23	39.83	5	27	68%	40.1	
Recursion	Binary to decimal	17	16.75	4	10	68%	42.3	
	Cross sum	12	14.66	3	4	84%	27.6	
	Factorial	12	16.50	3	4	95%	22.3	
	Fibonacci variation	12	10.88	3	4	84%	35.6	
	Power	17	15.38	4	8	79%	33.3	
If/Else	Contains yes or no	22	6.13	6	7	95%	23.4	
	Hurricane check	17	9.00	7	13	100%	21.5	
	Sort four elements	17	30.30	7	61	79%	41.4	

[1]https://github.com/BasLeijdekkers/MetricsReloaded/ [2]https://www.sosy-lab.org/~dbeyer/DepDigger/

level of programming experience (6.79 ± 4.96 years) and sufficient experience with object-oriented programming (self-estimated level of 2.83 ± 0.53 on a Likert scale of 1 to 5) and Java (self-estimated level of 3.47 ± 1.09 on a Likert scale of 1 to 5). Thus, the participants are intermediate programmers according to Dreyfus' taxonomy of skill acquisition [38], [39]. All participants had normal or corrected-to-normal vision and were right-handed. They received monetary compensation and could abort the experiment at any time. The study was approved by the local ethics board.

3) Design and Tasks: Our study design builds on a previous fMRI study [27]. We presented three tasks in the fMRI scanner. First, participants should comprehend a code snippet. To this end, they should determine what would be printed on the screen if the snippet was executed. When they responded, the experiment moved on, but there was an upper limit of 60 seconds for a comprehension task. A 10-second distractor task[5] followed to reduce reflective thoughts [41]. Finally, a 30-second rest condition followed. After three comprehension snippets, a control condition followed, in which participants saw another snippet and should click whenever they spotted an opening bracket. This process was repeated until the participant completed all 16 comprehension snippets. In Figure 1, we illustrate one out of five experiment trials.

4) Data Collection: We carried out the imaging sessions, which lasted around 45 minutes, on a 3-Tesla fMRI scanner[6] equipped with a 32-channel head coil. The heads of participants were fixed with a cushion with attached earmuffs containing fMRI-compatible headphones.[7] Participants wore earplugs for further noise protection.

We obtained a T1-weighted anatomical scan of the participants' brain with 1 mm isotropic resolution. We captured a whole-head fMRI using a continuous multi-band echo planar imaging (EPI) sequence with a dynamic number of volumes, as it depended on participant performance (echo time [TE]: 30 ms; repetition time [TR]: 1200 ms; flip angle [FA]: 60°; multi-band acceleration factor: 2; 36 slices of 3 mm thickness with 0.3 mm gaps).

During the fMRI session, we collected behavioral data with an fMRI-compatible two-button response device. Participants indicated whether they could compute the result of a snippet. We showed a warning after 58 seconds that the time was almost up.

After the fMRI session, we conducted a semi-structured interview with each participant, which was based on the results of the pilot studies. In addition to open-ended questions investigating the participants' individual perception of snippet complexity, we showed them the code snippets again and asked them to order the snippets regarding complexity. Such categorization tasks can produce insights into how participants

[5]We used the d2 task, a psychological test of attention in which participants scan through a row of letters and decide for each letter whether it is a *d* with two marks [40].

[6]Philips Achieva dStream, Best, The Netherlands
[7]MR Confon GmbH, Magdeburg, Germany

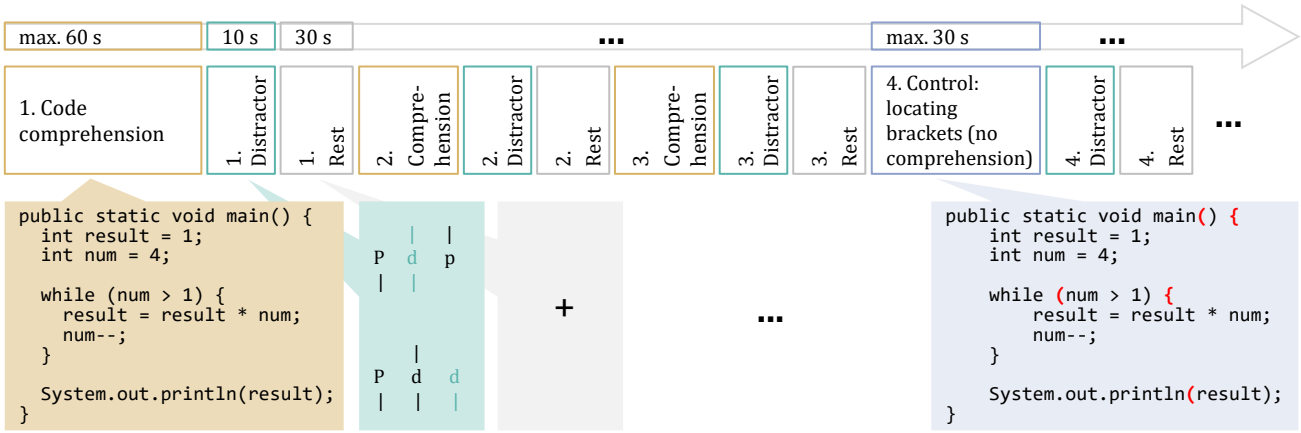

Fig. 1: Illustration of one (out of five) experiment trials.

approach a task (e.g., physics novices and experts categorize problems based on different aspects [42]). We allowed participants to self-choose the number of categories because we noticed in the pilot runs that participants had difficulties when a fixed number of piles did not match their expectation of different complexity levels. As in the pilot studies, this helps us to understand what made a snippet difficult or easy to comprehend. If participants made only two piles ("simple" and "complex"), we encouraged them to distinguish it further, often leading to three or four piles (see Section III-C).

5) Data Analysis: While our fMRI study of code complexity is novel, the fMRI data analysis represents best practices from neuroscience and follows previous studies in software engineering [27], [28], [43], [44], [45]. To analyze the fMRI data, we used BrainVoyager™ QX 2.8.4. We transformed each participant's anatomical scan into the standard Talairach brain [46], which allows us to correct for differences between the participants' brain sizes. We preprocessed the fMRI data with standard 3D-motion correction, slice-scan-time correction, temporal filtering (high-pass GLM Fourier, 2 cycles), and spatial smoothing (Gaussian filter, FWHM=4 mm).

We conducted a random-effects general linear model (GLM) analysis, defining one predictor per condition: program comprehension, distraction, control, and rest. To identify brain areas related to program comprehension, we filtered the data to all voxels that showed a positive deflection of the BOLD response during the comprehension condition. We computed the contrast between program comprehension and control condition ($p < 0.05$, false discovery rate (FDR) corrected [47], minimum cluster size: 27 mm^3). To identify deactivated brain areas, we obtained all voxels that show a negative BOLD deflection when contrasting comprehension and rest ($p < 0.001$, FDR corrected, minimum cluster size: 27 mm^3). For correlation analysis, we computed the mean amplitude of positive or negative BOLD deflection in percent for each cluster, task snippet, and participant, and correlated with the complexity metrics.

In addition to brain activation, we collected two further dependent variables: First, we observed participant behavior (response time, response correctness). This helped us to evaluate whether participants actually worked on understanding the snippets. We excluded snippets where participants (accidentally) responded too fast (response time less than 3 seconds), which happened in 3 out of 304 comprehension tasks. Second, we recorded subjective complexity in a post-interview: We converted the piles into equidistant values between 0 and 100. For example, if a participant created 4 piles, we assigned snippets of the first pile a complexity score of 0, the second pile a score of 33, the third 66, and the fourth 100. 3 piles translate to 0, 50, and 100, and 2 piles to 0 and 100.

To analyze the relationship between complexity metrics and cognitive processes, we use Kendall's τ to compare each metric's complexity value with the observed brain (de)activation across all snippets. We use Kendall's τ (rather than Pearson's correlation coefficient) because of its power with interval data and robustness against outliers [48].

III. Results

In this section, we present the results of our data analysis, including behavioral, fMRI, and subjective complexity data. We summarize the correlation results in Table II. We separate results from discussion (see Section IV) to prevent mixing interpretation with data. To streamline the presentation, we concentrate here on the four representative metrics. In Section IV-A3, we consider further metrics providing more evidence on the link between complexity metrics and cognitive processes.

A. Complexity Metrics and Behavioral data

On average, participants needed 32 seconds to solve a task, and solved 72 % of the tasks correctly. All participants were able to complete all tasks before the maximum experiment time was reached.

Regarding the relationship to complexity metrics, McCabe has no correlation to neither response time nor correctness. LOC, Halstead, and DepDegree all show a small correlation with response time and a medium correlation with correctness.

TABLE II: Kendall's τ and the explained variance (r^2, in brackets) of the dependent variables. A darker cell shading indicates a stronger correlation: none ($\tau < 0.1$), small ($0.1 < \tau < 0.3$), medium ($0.3 < \tau < 0.5$), and strong ($0.5 < \tau$) [49].

		Complexity Metrics				Activation				Deactivation		Subjective
		LOC	Halstead	McCabe	DepDegree	BA 6	BA 21	BA 39	BA 44/45	BA 31	BA 32	Complexity
Complexity Metrics	LOC											.16 (.04)
	Halstead	.32										.20 (.04)
	McCabe	.57	.25									−.07 (.01)
	DepDegree	.59	.50	.49								.16 (.01)
Activation	BA 6	.26 (.05)	.38 (.20)	.04 (.00)	.32 (.11)							.20 (.09)
	BA 21	.43 (.39)	.32 (.27)	.09 (.04)	.41 (.24)							.18 (.09)
	BA 39	.17 (.10)	.40 (.18)	.07 (.01)	.36 (.21)							.17 (.10)
	BA 44/45	.15 (.04)	.17 (.06)	−.04 (.00)	.22 (.09)							.22 (.06)
Deactivation	BA 31	−.30 (.21)	−.30 (.11)	.05 (.00)	−.24 (.04)							−.44 (.44)
	BA 32	−.39 (.24)	−.42 (.22)	.04 (.00)	−.29 (.04)							−.69 (.69)
Behavioral Data	Correctness	−.46 (.29)	−.45 (.26)	−.09 (.02)	−.41 (.22)	−.63 (.53)	−.63 (.67)	−.58 (.47)	−.34 (.13)	.59 (.58)	.71 (.62)	−.77 (.71)
	Time	.22 (.10)	.24 (.09)	.06 (.00)	.26 (.07)	.29 (.19)	.22 (.10)	.31 (.22)	.24 (.13)	−.22 (.11)	−.21 (.10)	.34 (.18)

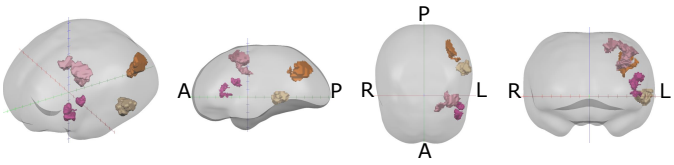

Fig. 2: Visualization of the activated brain areas (all in the left hemisphere): BA 6 (4315 voxel, TAL: -35, 10, 47), BA 21 (3306 voxel, TAL: -57, -39, 1), BA 39 (3527 voxel, TAL: -37, -65, 35), and BA 44/45 (Broca's area) (1618 voxel, TAL: -49, -22, 14). A: anterior, P: posterior, L: left, R: right.

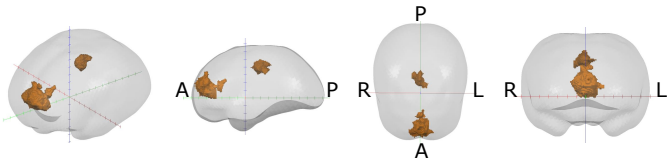

Fig. 3: Visualization of the deactivated brain areas: BA 32 (10'651 voxels, TAL: -1, 51, 14) and BA 31 (2672 voxels, TAL: 3, -20, 40), which both are part of the default mode network. A: anterior, P: posterior, L: left, R: right.

B. Complexity Metrics and fMRI Data

1) Brain Activation: In Fig. 2, we visualize the activated brain clusters. In our study, four brain areas are significantly activated during program comprehension: Brodmann area (BA)[8] 6, BA 21, BA 39, and BA 44/45 (Broca's area). Notably, these activation clusters were found also in previous fMRI studies of program comprehension [27], [28], [44], [51].

The relationship of the complexity metrics and activation strength of the four areas corroborates the behavioral data, but provides a stronger and more nuanced view (cf. Table II). Again, McCabe shows no correlation. LOC has a medium correlation with BA 21, and a small correlation with BA 6, BA 39, and BA 44/45. Halstead and DepDegree show a small correlation with BA 44/45 (cf. Fig. 4) and consistent medium correlations across BA 6, BA 21, and BA 39.

2) Brain Deactivation: Fig. 3 visualizes the specific positions of the two clusters that we found in BA 31 and BA 32, which belong to the default mode network. Regarding the relationship with the metrics, McCabe again shows no correlation with brain deactivation. DepDegree shows a small correlation that explains almost none of the observed variance in brain deactivation. LOC and Halstead have medium correlations with both deactivated areas (cf. Table II).

[8]Brodmann areas serve as an anatomical classification system. The entire brain is split into several areas on the basis of cytoarchitectonic differences, which suggest to serve different functional brain processes [50].

3) Behavioral Data and fMRI Data: We analyzed the relation of the behavioral data with the (de)activated areas: We found strong correlations with the response correctness for BA 6, BA 21, BA 39, and a medium correlation with BA 44/45. For response time, we found mostly small correlations, and one medium correlation (BA 39).

In summary, McCabe shows no correlation with the strength of brain activation or generic cognitive load. LOC, Halstead, and DepDegree show small to medium correlations with brain activation and cognitive load.

C. Subjective Complexity

In addition to objective measures of brain activation, we investigated the relationship between code complexity metrics and subjective complexity based on the participants' rating. Two participants created two piles (cf. Section II-C5), nine participants created three piles, and eight participants created four piles of complexity. Participants generally balanced the size of each pile. We transformed the piles into numerical values (42.42 ± 40.92). Regarding the relationship to the metrics, we observe only low correlations, with McCabe showing no correlation and the other metrics a small correlation only.

1) Subjective Complexity and Behavioral Data: We observe a medium correlation between response time and the subjective complexity rating. With the number of correctly solved tasks, the subjective rating shows the strongest correlation, in general.

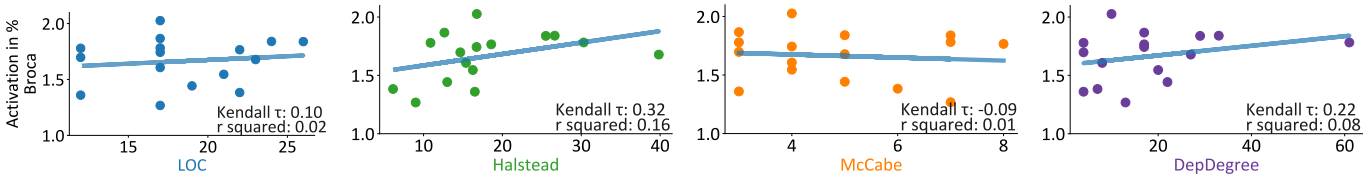

Fig. 4: Relationship of the four complexity metrics with Broca's area (BA 44/45). Each dot represents the mean activation for a single snippet. McCabe shows no correlation. LOC, Halstead, and DepDegree show small positive correlations, meaning that higher complexity values increase load in Broca's area. However, each metric only explains part of the observed variance in activation.

2) Subjective Complexity and fMRI Data: All (de)activated areas show mostly stronger correlations with the subjective complexity ratings than with the code complexity metrics. The deactivated areas of BA 31 and BA 32 show a medium and strong correlation, respectively. The four activated brain areas show only small correlations with subjective complexity.

In summary, while subjective complexity has small correlations with complexity metrics, it accurately depicts whether participants could solve a task. Subjective complexity also correlates with our measure of cognitive load.

IV. Discussion

We start the discussion by answering our overarching research question, followed by a detailed discussion of the relationship of code complexity metrics with cognitive processes. We conclude by formulating a set of hypotheses and outlining perspectives that arise from our study.

A. Overarching Research Question

Are complexity metrics connected to how programmers process code? Yes, and no. Based on plausibility and prior neuro-linguistic studies, we hypothesized some definitive relationships between code complexity metrics and programmers' cognition. For example, we expected that code with more data flow shows a direct positive correlation with BA 44/45 due to increased memory load [22]. While we indeed observe a positive correlation between DepDegree (as an indicator for data-flow complexity) and BA 44/45, we also observe small to medium-strength correlations with all other cognitive processing measures (i.e., all activated and deactivated brain areas and their levels) as well as behavioral data (i.e., response time and correctness). Similarly, LOC and Halstead exhibit small to medium-strength correlations with all behavioral and cognitive processing measures. McCabe, however, consistently lacked any significant correlation with our observed measures.

All of the observed relationships between complexity metrics and cognitive processing measures are nuanced and contextual, into which we delve next in more detail.

1) Deactivated Areas: Cognitive Load: When we must allocate attention to a task with perceived difficulty, areas of the brain associated with wandering and reflective thinking (default mode network) [36] are deactivated. The level of deactivation is an indicator for cognitive load [36]. Two classes of complexity metrics exhibited medium correlations with this deactivation: code size (LOC) and vocabulary (Halstead). In other words, the size of code, either as pure textual length or in terms of vocabulary size, drives cognitive load. We also observed a small correlation of data flow (DepDegree) but no correlation of control flow (McCabe) with deactivation. Perhaps, size-based metrics naturally interact with a programmer's sense of gestalt [52], and thus induce stronger anticipation for a higher cognitive load.

Our study results on complexity metrics substantiate our possibly spurious findings [26], but differ in a few key ways: We previously also observed a default mode network deactivation relationship with vocabulary size, but not code size. In contrast, now, with more varied snippets and complexity values, we found that code size also exhibits a medium correlation, in the same range as vocabulary size.

2) Activated Areas: Link to Cognitive Processes: When faced with solving a complex task, we perform additional cognitive operations (e.g., extracting the meaning of identifiers) and recruit additional cognitive resources and processes to fulfill the task's extra demands. The brain areas that are stronger activated in a complex task (as compared to a simple task) indicate increased demands for cognitive processes and resources hosted by these particular areas.

The LOC metric hints at an increased demand on a single area (as indicated by a medium-strength correlation), the middle temporal gyrus (BA 21). BA 21 is typically associated with semantic processing during language comprehension [53], [54] and program comprehension [27]. Its role for program comprehension is interpreted as extracting the meaning of individual identifiers and symbols from code [27], [28], [45], [44]. When processing complex sentences, increased activation of BA 21 indicates a higher grammatical processing load [55]. So, we conclude that merely increasing LOC increases the cognitive processing of identifiers and symbols, but otherwise does not necessarily pose a strong demand on other cognitive resources.

Halstead and DepDegree hint at increased demands (as indicated by medium-strength correlations) across three areas (BA 21, BA 6, and BA 39). The middle frontal gyrus (BA 6) is activated when attention and working memory are required [56]. Several fMRI studies on program comprehension found strong activation, albeit with slightly changing location in the brain [27], [28], [44]. The angular gyrus (BA 39) is a part of the brain that is associated with complex cognitive processes;

it acts as a hub for integrating incoming information. Like a reservoir, as processing areas are filled to capacity, other areas are recruited to share the load. Collectively, this result indicates that increasing the number of symbols and the number of data dependencies requires a broader network of processing than with increases in other factors. McCabe showed no relationship with an increased demand in any brain area.

Our results align well with the study by Schuster et al. [23], who found that an increasing number of words in a sentence leads to higher activation in BA 21. Furthermore, we found support for the early results of Curtis et al. [7] that program size relates to whether programmers successfully comprehend a piece of code. However, we did not find a link to McCabe, as Curtis et al. did.

3) Exploration of Further Complexity Metrics: So far, our analysis concentrated on commonly used representatives of each complexity metric class. However, while in widespread use, some of the selected representatives exhibit alleged limitations regarding human cognition. For example, McCabe's control-flow metric fails to consider the added complexity of nested structures, such as nested loops or recursion with complex break conditions. More recent control-flow metrics, such as "cognitive complexity" [57], take code repetition, layout, and modern program constructs into account and promise relief of such weaknesses.

To understand whether a more advanced (or any) metric predicts cognitive load better, we computed our snippets' complexity values of 43 further metrics provided by our analysis tools (i.e., MetricsReloaded and SONARQUBE). We included metrics that target the method level. Then, we excluded all complexity metrics that were unable to differentiate between our snippets (e.g., a metric counting the number of TODOs would yield 0 for all snippets and have no differentiating value for our analysis), which left 37 metrics, partially shown in Table III. We categorized the metrics into: 2 size metrics, 8 vocabulary metrics, 12 control-flow metrics, 0 data-flow metrics, and 15 others.

Overall, the 37 complexity metrics show a wide range of correlations with the observed brain activation. Interestingly, some simple control-flow metrics, such as the number of branching statements or the maximum loop depth, correlate more strongly than McCabe. SONARQUBE's "cognitive complexity" shows an improvement over McCabe, but only a small correlation, at best. This corroborates a prior meta-analysis on the limitations of the cognitive complexity regarding physiological data [58]. In addition to Halstead, the number of parameters is a second vocabulary-based metric that shows a strong correlation with brain activation in BA 44/45.

These findings corroborate that program comprehension is a complex cognitive process. While some simple metrics are well-suited to predict cognitive load in some brain areas, there is no single metric that predicts the overall cognitive effort. Advanced methods that try to capture all aspects are not an accurate predictor for cognitive load. Rather than trying to devise complex metrics with sophisticated and all-encompassing views on complexity, it may instead be worthwhile to use a basket of simple, but targeted metrics, with well-understood relationships with code and cognitive effort. For example, a continuous integration process could check simple metrics with well-understood cognitive relationships: maximum loop depth for constraints on programmers' working memory or the number of parameters as an indicator for load on semantic processing.

4) Summary: Considering the four representative complexity metrics, the vocabulary-size-based metric Halstead's complexity, followed by the data-flow-based metric DepDegree, has shown the most consistent relationship with the various cognitive effort measures. Control-flow complexity, as measured by McCabe, consistently lacked any relationship with cognitive effort. An exploration of other method-level metrics did not reveal any individual metric that accurately predicts cognitive load. However, besides DepDegree, none considered data-flow despite promising results. Future research shall consider data flow as a predictor for cognitive effort. While no single metric of complexity is sufficient for comprehensively explaining all observed data, we conclude that programmers should minimize the number of variables, branching depth, and amount of data flow within methods to reduce cognitive load when comprehending code.

Size-based metrics (Halstead and LOC) align well with the anticipation of work, and therefore with the amount of attention needed to allocate toward a task. Data-flow-based metrics (DepDegree) were less likely to capture this anticipation, yet demonstrated increased cognitive demands during program comprehension. These results imply that complexity estimates that use an assessment of the appearance of code, as is common in readability studies, rather than requiring actual comprehension of code, could misrepresent complexity.

Finally, a useful metric may involve the subjective complexity rating after the completion of a comprehension task. In other words, when participants struggled in grasping the meaning of a code snippet, they find it the most complex, and they seem to be well aware of their struggle. They are likely to predict their correctness ($\tau = -.77$), and their rating often matches their level of concentration ($\tau = -.69$), and relates to increases in cognitive effort.

B. Hypotheses

During the analysis and interviews, we found some further interesting insights that we formulate in terms of hypotheses to be addressed in future studies.

1) Size is a Preattentive Indicator for Cognitive Load: We found that, when a snippets consisted of more lines of code, the deactivation of the default mode network was rather strong. Thus, participants might simply use the amount of material to comprehend as a heuristic of how much cognitive load they expect. This is an easy to assess property, which might not even require attention and might be a feature of perception [52]. However, this could also lead to overestimation as the comprehension process progresses.

2) Data Flow versus Control Flow: Although DepDegree builds on McCabe, McCabe shows no relationship to cognitive effort, whereas DepDegree does. Thus, everything that

TABLE III: Kendall's τ correlation between brain activation and the unique, differentiating top 26 of the 37 explored complexity metrics. The metric's text color indicates its class (size, vocabulary, control flow, data flow, other). The cell shading highlights strong correlations (cf. Table II). All metrics, raw data, and results are available on our supplementary Web site.

Metric	BRANCH	NP	NBD	D	LOOP	EXP	N	V	CRCT	MAINT	iv(G)	RLOC	n	CALL	RLBTY	E	IF_NEST	CogCompl	CONTR	STAT	ev(G)	v(G)	EXEC	RETURN	CALLED	CDENS
BA 6	.50	.40	.36	.51	.36	.45	.41	.41	.41	.40	.34	.37	.37	.27	.34	−0.06	.31	.30	.27	.29	.23	.21	.16	−0.37	−0.18	.09
BA 21	.56	.27	.54	.41	.48	.34	.36	.36	.35	.33	.38	.37	.35	.37	.27	−0.07	.27	.24	.21	.23	.27	.10	.10	.10	.14	−0.33
BA 39	.29	.56	.29	.45	.22	.32	.34	.26	.31	.33	.10	.32	.17	.21	.27	−0.01	.31	.14	.13	.25	.19	.12	.19	−0.18	.00	−0.07
BA 44/45	.16	.50	.20	.41	.12	.16	.18	.14	.15	.17	.06	.16	.12	.16	.11	.03	.27	.04	.01	.07	.08	.00	.02	−0.08	.13	−0.05

distinguishes DepDegree from McCabe might be responsible for the small to medium correlations. Thus, a metric capturing only the part that differs between DepDegree and McCabe might be a good predictor for cognitive load. Thus, our results let us conclude that the delta between control flow measured by McCabe and data-flow measured by DepDegree may be a promising, novel metric that resembles programmers' cognition and shall be explored in future research.

C. Perspectives

1) Mental Shortcuts during Program Comprehension: Programmers tend to minimize efforts for program comprehension by actively looking for efficient ways to solve a task [59]. In the context of our experiment, this means that participants will try to find the simplest path to solve the task. For example, a participant reported, upon recognizing a list of square numbers, they expected an algorithm dealing with square roots (see top-down comprehension [60], [28]). Complexity metrics seem to neglect that programmers try to take such mental shortcuts. For example, McCabe counts the number of *all* possible execution paths, but programmers often do not have to consider all paths; just enough to solve the current task. For example, in our "hurricane check" snippet, participants could skip some `if` statements after they have found the solution. Further work shall investigate this phenomenon by carefully controlling which mental shortcuts during comprehension are available. This would produce further insights into the context sensitivity of complexity metrics.

2) The Effects of Extreme Complexity: Some metrics may not have a relationship with human cognition, until the values exceed some extreme threshold. Although a higher number of possible execution paths did not increase the cognitive load of participants, there are only 6 different values for McCabe, and the largest (i.e., 8) might not even be considered sufficiently complex. Still, we selected them for their widespread use in practice. Some static analysis tools, such as SonarQube, consider a McCabe value of 10 to be so complex, that it should not even be checked into a repository. Unfortunately, it is almost impossible to vary the code along all 4 classes of metrics while also adhering to presenting the full code on one screen within the fMRI scanner. Thus, future studies could either use longer code, which would require the participants to scroll up and down, or could focus more on control flow to have more extreme ranges so that the relationship of complexity and cognitive load can be observed in more depth.

3) Activation of Broca for High Performers: We confirmed the activation of the BA 44/45 (Broca's area), which is crucially involved in establishing a unified understanding between alternatives (e.g., combining the meaning of words to a sentence) [61] and which was consistently activated in previous fMRI studies on program comprehension [26], [28], [51] and sentence complexity [55].

Correlations between our complexity metrics and the observed activation in Broca's area are mostly small. One reason could be that activation of Broca's area was modulated by individual performance. That is, some participants, finding the code too complex, fail to activate later stages of language processing. Another possibility could be that the code snippets were more mentally challenging than in previous studies, and as a result, some participants needed to recruit Broca's area as syntactic working memory [22]. Both explanations are consistent with our data, as we found stronger activation in Broca's area among the *high-performing participants* (i.e., who correctly solved, at least, 13 of the 16 comprehension tasks, $n = 9$). Studies examining complex sentence comprehension of individuals have also observed activation differences between high-performing individuals with good comprehension (increased activation of Broca's area) and poor comprehension (decreased activation of Broca's area) [62].

Future experiments shall examine when and why Broca's area is activated by high performers, increasing our understanding of expertise. Furthermore, if we want code to be understandable by *everyone*, then we can use these methods to design metrics that predict truly simple code, code that does not require the brain circuity associated with complexity.

4) Overcoming Shortcomings of Complexity Metrics: Based on the participants' feedback and the observed relationships to various measures of programmer cognition, we found that popular complexity metrics fail to capture some comprehension aspects that participants used in their subjective rating:

- Long, diffuse code lines can cause particular difficulties. For example, Line 11 of Listing 1 obfuscates the intention (is it a non-letter?), which needs to be extracted from the code.
- Identifiers with similar names lead to confusion (e.g., `number1`, `number2`, `numbers` in one snippet), likely because participants have to pay specific attention not

to confuse these. Thus, code *readability* may be just as important as structural complexity, as suggested in previous studies [63], [64], [65].
- DepDegree fails to consider the "distance" or the locality of data-flow relationships. For example, in the `sort` example in the prelude of the paper, the swap operations are localized to each line and can be abstracted away once a line has been processed. Incorporating other factors, such as the variable lifetime or lexical distance, would be worthwhile to explore.

These and other aspects shall be considered when using code complexity metrics to describe human cognition (e.g., Jbara and Feitelson consider repeated statements [18]). Our experiment design provides a structured way to test and refine code complexity metrics to make them a more accurate proxy for program comprehension and elevate them beyond simple code size predictors [66].

V. Threats to Validity

A. Construct Validity

We carefully designed our experiment to limit threats to validity. With regard to construct validity, we operationalized code complexity with four widely used metrics covering different concepts of code complexity (e.g., control-flow or data-flow complexity). An exploration into 37 further metrics did not reveal any candidates with consistently strong correlations. The four selected metrics correlate with each other even with a carefully designed experiment. Nevertheless, in addition to our conceptual insights, our study outlined how to investigate a possible *cognitive* complexity metric with a multi-modal experiment.

Another threat arises from our operationalization of program comprehension, which is a multi-faceted phenomenon in which the chosen experiment task is decisive for observed cognitive processes [67]. In our study, we asked programmers to evaluate snippets regarding input and output, and we found, at most, a medium correlation with code complexity metrics. An experiment with another type of task (e.g., deriving program invariants) may emphasize a different facet of program comprehension and thus may show stronger or weaker correlations with complexity metrics. However, our task operationalization is typical to specify program behavior and in line with previous fMRI studies on program comprehension [27], [28], [45], [43], [44]. While no single metric of complexity is sufficient for comprehensively explaining all observed data, we can conclude that programmers should minimize the number of variables, branching depth, and amount of data flow within methods to reduce cognitive load when comprehending the code.

B. Internal and External Validity

Several threats to validity arise from our participant sample.

Participants: First, we have a skewed gender distribution, which, however, is close to the population in computer science for most universities. Second, participants may have encountered algorithms used in our snippets before. However, we mitigated this threat by enforcing bottom-up comprehension.

Code Snippet Selection: Due to the nature of controlled fMRI experiments, we intentionally focused on high internal validity to control for confounding parameters as much as possible. Our snippets are rather small, in one programming language, and we selected a homogeneous sample in terms of programming experience. Thus, our results apply only to similar circumstances and cannot easily be generalized, for example, to expert programmers or large code bases. This is an unavoidable trade-off between targeting either high internal or high external validity [68].

Code Complexity Granularity: We need to be aware that, in our experiment, we studied program comprehension at the method level, but software systems consist of many methods and higher-level components. Nevertheless, our results still have practical impact: When we know that intermediate programmers work at the method level, code complexity metrics can help to predict their cognitive effort and that they might need longer than expected. Furthermore, different complexity metrics have been devised beyond the method level (e.g., Weighted Methods per Class [69], Lack of Cohesion in Methods [6], [70]), which shall be addressed in future research. Our study provides a starting point for dedicated follow-up studies that shall investigate these metrics and associated cognitive processes.

VI. Related Work

Besides the work that evaluates how software metrics are related to human cognition (cf. Section I), several neuroimaging studies exist that shed light on how programmers work with code. Closest to our study are studies on cognitive load and neural efficiency. Siegmund et al. conducted an fMRI study and found that the strength of activation depends on expertise [28]. Specifically, top-down comprehension resulted in a lower BOLD signal change than bottom-up comprehension. Crk and Kluthe found different strengths in electroencephalography (EEG) alpha and theta power depending on the expertise of participants [71]. Yeh et al. also observed a difference in alpha and theta waves, linked to the cognitive load of participants [72]. Both results suggest that cognitive load and expertise are directly linked (i.e., with lower expertise, the same tasks require higher cognitive load). Nakagawa et al. found an increased blood flow with functional near-infrared spectroscopy (fNIRS) in the prefrontal cortex depending on task difficulty [73]. Fakhoury et al. combined fNIRS and eye tracking to show that unsuitable identifier names increase cognitive load [74]. Kosti et al. replicated the seminal fMRI study of Siegmund et al. [27] with EEG and confirmed its capability as a mental load measure by correlating the EEG functional connectivity with subjective rating of difficulty [75]. Fritz et al. used EEG, heart rate, and electrodermal activity to successfully predict task difficulty [76]. Ikutani et al. used fMRI to contrast different levels of programmer expertise with a program categorization task. Their classifier achieved a higher accuracy of distinguishing different program categories on expert programmers' brain activation patterns than for novices, indicating that expertise leads to a fine-tuning of programmers' brains [51]. The results of these studies motivated us to keep

the level of programming experience constant and to focus on bottom-up comprehension.

Siegmund et al. have brought this line of research into software engineering with a study to unraveling the neuronal correlates of program comprehension [27]. A follow-up paper, which includes a protocol to link software metrics to cognitive load, has directly inspired this work [26]. Peitek et al. suggested that neuronal correlates can be interpreted in more detail by simultaneously observing eye gaze [41]. Floyd et al. conducted a study to predict the tasks that participants completed based on the observed brain activation [45]. They could successfully predict whether participants comprehended code, reviewed a code change, or reviewed a prose change. Follow-up studies investigated mental rotations tasks and contrasted the use of fMRI with fNIRS [77]; they found that brain activation is much different when writing code, rather than comprehending it [78]. Lee et al. also built on the first fMRI study, but used EEG and eye tracking to describe neuronal correlates of program comprehension in two studies [79], [80]. Duraes et al. and Castelhano et al. used fMRI to observe the neuronal activation during the location of defects, finding an activation in the right anterior insula when a bug was spotted and confirmed [43], [44]. Ikutani and Uwano used fNIRS and found an increased activation in the frontal pole when participants memorized variable names, as compared to mental arithmetic [81]. Two recent studies by Liu et al. and Ivanovo et al. contrasted program comprehension with language and memory tasks and found activation in brain areas involved in formal logic as well as domain-general executive resources [82], [83]. While all these studies considered neuronal correlates of processes related to program comprehension, none establishes a link to software metrics.

VII. Conclusion

Code complexity metrics are relevant for researchers and practitioners alike, especially as a proxy for difficulty during comprehending code. Despite their widespread use, the validity of code complexity metrics is debated and, despite substantial research, the big picture is still unclear. To shed light on this issue, in an fMRI study, we investigated 41 complexity metrics and their behavioral and neuronal correlates during program comprehension. We found corroborating evidence with mostly weak to medium correlations with programmers' correctness and response time. More importantly, since we observed participants' brain activation with fMRI, we enriched previous research by offering a novel perspective and explaining *why* code and certain aspects of it are difficult to comprehend. In particular, we found that the code's textual size drives cognitive load due to programmers' expected attention and that vocabulary size of code particularly burdens programmers' working memory. The data-flow metric DepDegree showed the strongest correlations to our observed brain activation. We also observed that subjective complexity accurately mirrors the participants' need for concentration.

Despite these encouraging results, further studies shall dig deeper to better understand the suitability of code complexity metrics as a proxy for programmers' cognition. Data-flow-based metrics, such as DepDegree, showed promise and need further investigation. Future work shall also address the gap on how individual programmer behavior and knowledge enables mental shortcuts and thus reducing generic precision of complexity metrics.

Data Availability

Along with our study, we provide:

- a replication package, which includes all code snippets, their complexity metric values, and experiment scripts,
- results of our analyses for possible further hypotheses generation by other researchers, and
- an analysis pipeline that generates all correlations, statistics, and plots based on (our) input data.

Acknowledgment

We thank all participants of our pilot studies and the fMRI study. Furthermore, we thank Andreas Fügner, Anke Michalsky, and Jörg Stadler for their technical support during fMRI data acquisition, as well as Annabelle Bergum for her assistance with analyzing all 41 complexity metrics.

Apel's work has been supported by the German Research Foundation (AP 206/6). Brechmann's and Siegmund's work is supported by DFG grants BR 2267/7-2 and SI 2045/2-2. Siegmund's work is further funded by the Bavarian State Ministry of Education, Science and the Arts in the framework of the Centre Digitisation.Bavaria (ZD.B). Parnin's work is supported by the National Science Foundation under grant number 1755762.

References

[1] M. Codish, L. Cruz-Filipe, M. Nebel, and P. Schneider-Kamp, "Applying Sorting Networks to Synthesize Optimized Sorting Libraries," in *Proc. Int'l Symposium on Logic-Based Program Synthesis and Transformation*. Springer, 2015, pp. 127–142.

[2] A. S. N. Varela, H. G. Pérez-González, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, "Source Code Metrics: A Systematic Mapping Study," *J. Systems and Software*, vol. 128, pp. 164–197, 2017.

[3] M. Halstead, *Elements of Software Science*. Elsevier Science Inc., 1977.

[4] T. McCabe, "A Complexity Measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, 1976.

[5] D. Beyer and A. Fararooy, "A Simple and Effective Measure for Complex Low-Level Dependencies," in *Proc. Int'l Conf. Program Comprehension (ICPC)*. IEEE, 2010, pp. 80–83.

[6] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall, 1995.

[7] B. Curtis, S. B. Sheppard, P. Milliman, M. Borst, and T. Love, "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 2, pp. 96–104, 1979.

[8] H. Zuse, "Criteria for Program Comprehension Derived from Software Complexity Metrics," in *Workshop on Program Comprehension*. IEEE, 1993, pp. 8–16.

[9] H. Sneed, "Understanding Software Through Numbers: A Metric Based Approach to Program Comprehension," *Journal of Software Maintenance: Research and Practice*, vol. 7, no. 6, pp. 405–419, 1995.

[10] N. Fenton and S. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 3rd ed. CRC Press, 2014.

[11] J. García-Munoz, M. García-Valls, and J. Escribano-Barreno, "Improved Metrics Handling in SonarQube for Software Quality Monitoring," in *Proc. Int'l Conf. Distributed Computing and Artificial Intelligence*. Springer, 2016, pp. 463–470.

[12] L. Ottenstein, V. Schneider, and M. Halstead, "Predicting the Number of Bugs Expected in a Program Module," Department of Computer Science, University of Purdue, Tech. Rep. 76-205, 1976.

[13] R. Moser, W. Pedrycz, and G. Succi, "A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction," in *Proc. Int'l Conf. Software Engineering (ICSE)*, 2008, pp. 181–190.

[14] S. Huda, S. Alyahya, M. M. Ali, S. Ahmad, J. Abawajy, H. Al-Dossari, and J. Yearwood, "A Framework for Software Defect Prediction and Metric Selection," *IEEE Access*, vol. 6, pp. 2844–2858, 2017.

[15] C. Kaner and W. P. Bond, "Software Engineering Metrics: What Do They Measure and How Do We Know?" in *Proc. Int'l Software Metrics Symposium (METRICS)*. IEEE, 2004, p. 12.

[16] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto, "Automatically Assessing Code Understandability: How Far Are We?" in *Proc. Int'l Conf. Automated Software Engineering (ASE)*. IEEE, 2017, pp. 417–427.

[17] S. Ajami, Y. Woodbridge, and D. Feitelson, "Syntax, Predicates, Idioms - What Really Affects Code Complexity?" in *Proc. Int'l Conf. Program Comprehension (ICPC)*. IEEE, 2017, pp. 66–76.

[18] A. Jbara and D. Feitelson, "How Programmers Read Regular Code: A Controlled Experiment Using Eye Tracking," *Empirical Softw. Eng.*, vol. 22, no. 3, pp. 1440–1477, 2017.

[19] A. Caramazza and E. B. Zurif, "Dissociation of Algorithmic and Heuristic Processes in Language Comprehension: Evidence from Aphasia," *Brain and Language*, vol. 3, no. 4, pp. 572–582, 1976.

[20] J. Uddén, A. Hultén, J.-M. Schoffelen, N. Lam, K. Harbusch, A. van den Bosch, G. Kempen, K. M. Petersson, and P. Hagoort, "Supramodal Sentence Processing in the Human Brain: fMRI Evidence for the Influence of Syntactic Complexity in More Than 200 Participants," *bioRxiv*, p. 576769, 2019.

[21] A. Santi and Y. Grodzinsky, "Working Memory and Syntax Interact in Broca's Area," *Neuroimage*, vol. 37, no. 1, pp. 8–17, 2007.

[22] C. J. Fiebach, M. Schlesewsky, and A. D. Friederici, "Syntactic Working Memory and the Establishment of Filler-Gap Dependencies: Insights from ERPs and fMRI," *Journal of Psycholinguistic Research*, vol. 30, no. 3, pp. 321–338, 2001.

[23] S. Schuster, S. Hawelka, N. A. Himmelstoss, F. Richlan, and F. Hutzler, "The Neural Correlates of Word Position and Lexical Predictability during Sentence Reading: Evidence from Fixation-Related fMRI," *Language, Cognition and Neuroscience*, vol. 35, no. 5, pp. 613–624, 2020.

[24] J. Liu, M. Zhang, J. Jou, X. Wu, W. Li, and J. Qiu, "Neural Bases of Falsification in Conditional Proposition Testing: Evidence from an fMRI Study," *Int'l Journal of Psychophysiology*, vol. 85, no. 2, pp. 249–256, 2012.

[25] E. Kulakova, M. Aichhorn, M. Schurz, M. Kronbichler, and J. Perner, "Processing Counterfactual and Hypothetical Conditionals: An fMRI Investigation," *NeuroImage*, vol. 72, pp. 265–271, 2013.

[26] N. Peitek, J. Siegmund, S. Apel, C. Kästner, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "A Look into Programmers' Heads," *IEEE Trans. Softw. Eng.*, vol. 46, no. 4, pp. 442–462, 2020.

[27] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "Understanding Understanding Source Code with Functional Magnetic Resonance Imaging," in *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 2014, pp. 378–389.

[28] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, and A. Brechmann, "Measuring Neural Efficiency of Program Comprehension," in *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 2017, pp. 140–150.

[29] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm, "Eye Movements in Code Reading: Relaxing the Linear Order," in *Proc. Int'l Conf. Program Comprehension (ICPC)*. IEEE, 2015, pp. 255–265.

[30] N. Pennington, "Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs," *Cognitive Psychology*, vol. 19, no. 3, pp. 295–341, 1987.

[31] R. Brooks, "Towards a Theory of the Comprehension of Computer Programs," *J. Man-Machine Studies*, vol. 18, no. 6, pp. 543–554, 1983.

[32] R. Hoge and B. Pike, "Quantitive Measurement Using fMRI," in *Functional Magnetic Resonance Imaging: An Introduction to Methods*. Oxford University Press, 2001, pp. 159–174.

[33] M. S. Gazzaniga, R. B. Ivry, and G. R. Mangun, *Cognitive Neuroscience: The Biology of the Mind*. Norton & Company, 2013.

[34] M. Raichle, A. MacLeod, A. Snyder, W. Powers, D. Gusnard, and G. Shulman, "A Default Mode of Brain Function," *Proc. Nat'l Academy of Sciences*, vol. 98, no. 2, pp. 676–682, 2001.

[35] L. Mak, L. Minuzzi, G. MacQueen, G. Hall, S. Kennedy, and R. Milev, "The Default Mode Network in Healthy Individuals: A Systematic Review and Meta-Analysis," *Brain Connectivity*, vol. 7, no. 1, pp. 25–33, 2017.

[36] K. McKiernan, J. Kaufman, J. Kucera-Thompson, and J. Binder, "A Parametric Manipulation of Factors Affecting Task-Induced Deactivation in Functional Neuroimaging," *J. Cognitive Neuroscience*, vol. 15, no. 3, pp. 394–408, 2003.

[37] J. Siegmund, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg, "Measuring and Modeling Programming Experience," *Empirical Softw. Eng.*, vol. 19, no. 5, pp. 1299–1334, 2014.

[38] H. Dreyfus and S. Dreyfus, "Mind over machine: The power of human intuition and expertise in the era of the computer," *The Free Press*, 1986.

[39] J. Mead, S. Gray, J. Hamer, R. James, J. Sorva, C. S. Clair, and L. Thomas, "A Cognitive Approach to Identifying Measurable Milestones for Programming Skill Acquisition," *ACM SIGCSE Bulletin*, vol. 38, no. 4, pp. 182–194, 2006.

[40] R. Brickenkamp, L. Schmidt-Atzert, and D. Liepmann, *Test d2-Revision: Aufmerksamkeits-und Konzentrationstest*. Hogrefe Göttingen, 2010.

[41] N. Peitek, J. Siegmund, C. Parnin, S. Apel, J. Hofmeister, and A. Brechmann, "Simultaneous Measurement of Program Comprehension with fMRI and Eye Tracking: A Case Study," in *Proc. Int'l Symp. Empirical Software Engineering and Measurement (ESEM)*. ACM, 2018, pp. 24:1–24:10.

[42] J. Snyder, "An Investigation of the Knowledge Structures of Experts, Intermediates and Novices in Physics," *Science Education*, vol. 22, no. 9, pp. 979–992, 2000.

[43] J. Duraes, H. Madeira, J. Castelhano, I. C. Duarte, and M. Castelo-Branco, "WAP: Understanding the Brain at Software Debugging," in *Proc. Int'l Symposium Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 87–92.

[44] J. Castelhano, I. Duarte, C. Ferreira, J. Duraes, H. Madeira, and M. Castelo-Branco, "The Role of the Insula in Intuitive Expert Bug Detection in Computer Code: An fMRI Study," *Brain Imaging and Behavior*, vol. 13, no. 3, pp. 1–15, May 2018.

[45] B. Floyd, T. Santander, and W. Weimer, "Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise," in *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE, 2017, pp. 175–186.

[46] J. Talairach and P. Tournoux, *Co-Planar Stereotaxic Atlas of the Human Brain*. Thieme, 1988.

[47] Y. Benjamini and Y. Hochberg, "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing," *J. Royal Statistical Society. Series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.

[48] J. Cohen and P. Cohen, *Applied Multiple Regression: Correlation Analysis for the Behavioral Sciences*, 2nd ed. Addison Wesley, 1983.

[49] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Academic Press, 1969.

[50] K. Brodmann, *Brodmann's Localisation in the Cerebral Cortex*. Springer, 2006.

[51] Y. Ikutani, T. Kubo, S. Nishida, H. Hata, K. Matsumoto, K. Ikeda, and S. Nishimoto, "Expert Programmers have Fine-Tuned Cortical Representations of Source Code," *bioRxiv*, 2020.

[52] B. Goldstein, *Sensation and Perception*, 5th ed. Cengage Learning Services, 2002.

[53] J. Binder, R. Desai, W. Graves, and L. Conant, "Where Is the Semantic System? A Critical Review and Meta-Analysis of 120 Functional Neuroimaging Studies," *Cerebral Cortex*, vol. 19, no. 12, pp. 2767–2796, 2009.

[54] N. Dronkers, D. Wilkins, R. Van Valin Jr, B. Redfern, and J. Jaeger, "Lesion Analysis of the Brain Areas Involved in Language Comprehension," *Cognition*, vol. 92, no. 1–2, pp. 145–177, 2004.

[55] E. Kaan and T. Swaab, "The Brain Circuitry of Syntactic Comprehension," *Trends in Cognitive Sciences*, vol. 6, no. 8, pp. 350–356, 2002.

[56] K. Nebel, H. Wiese, P. Stude, A. de Greiff, H.-C. Diener, and M. Keidel, "On the Neural Basis of Focused and Divided Attention," *Cognitive Brain Research*, vol. 25, no. 3, pp. 760–776, 2005.

[57] A. Campbell, "Cognitive Complexity: A New Way of Measuring Understandability," SonarSource S.A., Tech. Rep., 2018.

[58] M. M. Barón, M. Wyrich, and S. Wagner, "An Empirical Validation of Cognitive Complexity as a Measure of Source Code Understandability,"

in *Proc. Int'l Symp. Empirical Software Engineering and Measurement (ESEM)*, 2020, p. 12.

[59] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How Do Professional Developers Comprehend Software?" in *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE, 2012, pp. 255–265.

[60] R. Brooks, "Using a Behavioral Theory of Program Comprehension in Software Engineering," in *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE, 1978, pp. 196–201.

[61] P. Hagoort, "On Broca, Brain, and Binding: A New Framework," *Trends in Cognitive Sciences*, vol. 9, no. 9, pp. 416–423, 2005.

[62] H. Van Ettinger-Veenstra, A. McAllister, P. Lundberg, T. Karlsson, and M. Engström, "Higher Language Ability is Related to Angular Gyrus Activation Increase During Semantic Processing, Independent of Sentence Incongruency," *Frontiers in Human Neuroscience*, vol. 10, p. 110, 2016.

[63] R. Buse and W. Weimer, "A Metric for Software Readability," in *Proc. Int'l Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2008, pp. 121–130.

[64] ——, "Learning a Metric for Code Readability," *IEEE Trans. Softw. Eng.*, vol. 36, no. 4, pp. 546–558, 2010.

[65] D. Posnett, A. Hindle, and P. Devanbu, "A Simpler Model of Software Readability," in *Proc. Int'l Conf. Mining Software Repositories (MSR)*. ACM, 2011, pp. 73–82.

[66] Y. Gil and G. Lalouche, "On the Correlation between Size and Metric Validity," *Empirical Softw. Eng.*, vol. 22, no. 5, pp. 2585–2611, 2017.

[67] A. Dunsmore and M. Roper, "A Comparative Evaluation of Program Comprehension Measures," *J. Systems and Software*, vol. 52, no. 3, pp. 121–129, 2000.

[68] J. Siegmund, N. Siegmund, and S. Apel, "Views on Internal and External Validity in Empirical Software Engineering," in *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE, 2015, pp. 9–19.

[69] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.

[70] M. Hitz and B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented Systems," in *Proc. Int'l Symp. Applied Corporate Computing*, 1995, pp. 299–300.

[71] I. Crk and T. Kluthe, "Toward Using Alpha and Theta Brain Waves to Quantify Programmer Expertise," in *Proc. Int'l Conf. Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2014, pp. 5373–5376.

[72] M. Yeh, D. Gopstein, Y. Yan, and Y. Zhuang, "Detecting and Comparing Brain Activity in Short Program Comprehension Using EEG," in *Frontiers in Education Conference*. IEEE, 2017, pp. 1–5.

[73] T. Nakagawa, Y. Kamei, H. Uwano, A. Monden, K. Matsumoto, and D. M. German, "Quantifying Programmers' Mental Workload During Program Comprehension Based on Cerebral Blood Flow Measurement: A Controlled Experiment," in *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 2014, pp. 448–451.

[74] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load," in *Proc. Int'l Conf. Program Comprehension (ICPC)*. IEEE, 2018, pp. 286–296.

[75] M. Kosti, K. Georgiadis, D. Adamos, N. Laskaris, D. Spinellis, and L. Angelis, "Towards an Affordable Brain Computer Interface for the Assessment of Programmers' Mental Workload," *Int'l J. Human-Computer Studies*, vol. 115, pp. 52–66, 2018.

[76] T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliott, and M. Züger, "Using Psycho-Physiological Measures to Assess Task Difficulty in Software Development," in *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 2014, pp. 402–413.

[77] Y. Huang, X. Liu, R. Krueger, T. Santander, X. Hu, K. Leach, and W. Weimer, "Distilling Neural Representations of Data Structure Manipulation using fMRI and fNIRS," in *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE, 2019, pp. 396–407.

[78] R. Krueger, Y. Huang, X. Liu, T. Santander, W. Weimer, and K. Leach, "Neurological Divide: An fMRI Study of Prose and Code Writing," in *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE, 2020, pp. 678–690.

[79] S. Lee, A. Matteson, D. Hooshyar, S. Kim, J. Jung, G. Nam, and H. Lim, "Comparing Programming Language Comprehension between Novice and Expert Programmers Using EEG Analysis," in *Int'l Conf. on Bioinformatics and Bioengineering (BIBE)*. IEEE, 2016, pp. 350–355.

[80] S. Lee, D. Hooshyar, H. Ji, K. Nam, and H. Lim, "Mining Biometric Data to Predict Programmer Expertise and Task Difficulty," *Cluster Computing*, vol. 21, no. 1, pp. 1097–1107, 2017.

[81] Y. Ikutani and H. Uwano, "Brain Activity Measurement during Program Comprehension with NIRS," in *Int'l Conf. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2014, pp. 1–6.

[82] Y.-F. Liu, J. Kim, C. Wilson, and M. Bedny, "Computer Code Comprehension Shares Neural Resources With Formal Logical Inference in the Fronto-Parietal Network," *bioRxiv*, 2020.

[83] A. Ivanova, S. Srikant, Y. Sueoka, H. Kean, R. Dhamala, U.-M. O'Reilly, M. Bers, and E. Fedorenko, "Comprehension of Computer Code Relies Primarily on Domain-General Executive Resources," *bioRxiv*, 2020.