Towards a Cognitive Model of Dynamic Debugging: Does Identifier Construction Matter?

Danniell Hu, Priscila Santiesteban, Madeline Endres, Westley Weimer

Abstract—Debugging is a vital and time-consuming process in software engineering. Recently, researchers have begun using neuroimaging to understand the cognitive bases of programming tasks by measuring patterns of neural activity. While exciting, prior studies have only examined small sub-steps in isolation, such as comprehending a method without writing any code or writing a method from scratch without reading any already-existing code. We propose a simple multi-stage debugging model in which programmers transition between Task Comprehension, Fault Localization, Code Editing, Compiling, and Output Comprehension activities. We conduct a human study of $n = 28$ participants using a combination of functional near-infrared spectroscopy and standard coding measurements (e.g., time taken, tests passed, etc.). Critically, we find that our proposed debugging stages are both neurally and behaviorally distinct. To the best of our knowledge, this is the first neurally-justified cognitive model of debugging.

At the same time, there is significant interest in understanding how programmers from different backgrounds, such as those grappling with challenges in English prose comprehension, are impacted by code features when debugging. We use our cognitive model of debugging to investigate the role of one such feature: identifier construction. Specifically, we investigate how features of identifier construction impact neural activity while debugging by participants with and without reading difficulties. While we find significant differences in cognitive load as a function of morphology and expertise, we do not find significant differences in end-to-end programming outcomes (e.g., time, correctness, etc.). This nuanced result suggests that prior findings on the cognitive importance of identifier naming in isolated sub-steps may not generalize to end-to-end debugging. Finally, in a result relevant to broadening participation in computing, we find no behavioral outcome differences for participants with reading difficulties.

Index Terms—software engineering, debugging, fNIRS, reading ability, cognitive processes, identifier naming

1 Introduction

Software debugging is the multi-step process of finding and fixing bugs in code. Debugging is a critical activity within software engineering, consuming 35–50% of developers' time [1]. Despite advances in development tools designed to prevent or fix errors in code [2], [3], debugging remains a programmer-driven process. Understanding the cognition of debugging could inform both debugging interventions and practices, as well as lead to cognitively-informed debugging tools that better support developers.

Researchers are increasingly using neuroimaging to understand cognition during programming [4]. Such studies produce cognitive models of programming tasks by identifying patterns of neural activity. These studies have informed pedagogical interventions to improve novices' programming outcomes [5], [6] and helped broaden participation in computing by identifying if trainable skills (such as spatial ability [7], [8]) are relevant to programming.

Debugging-related programming tasks, such as program comprehension, code editing, and bug detection, have been separately studied using neuroimaging [9]–[13]. However, to the best of our knowledge, the cognition of debugging has not been studied as an integrated and dynamic process. This is important because debugging involves transitions between multiple interconnected programming tasks.

We desire a cognitive model of debugging that captures multiple debugging stages, is applicable to real-world code, and accounts for human differences. When a model generalizes to realistic code, it increases ecological validity, thereby improving practical applicability and facilitating better training and tool development [14]. In addition, modern software engineering involves developers from diverse neurological and linguistic backgrounds [15]. Psychological research has shown that individuals with reading difficulties (such as non-native speakers or people with dyslexia) can face pronounced challenges reading prose [16], [17]. It is crucial to investigate if such challenges extend to debugging.

We propose a dynamic debugging model that links software engineering activities (e.g., fault localization, code editing, etc.) to patterns of neural activity. We validate our model via an experimentally-controlled neuroimaging study of $n = 28$ programmers. To identify debugging stages, we captured programming keystrokes, window-switching, response accuracy, and response time data. To measure neural activity, we used functional near-infrared spectroscopy (fNIRS). Participants were asked to debug 13 Python Leetcode-style programs presented in Visual Studio Code, a standard IDE. While Leetcode problems may not be fully indicative of industrial development [18], they are commonly faced by junior developers entering the workforce [19]. Participants were given a problem prompt, defect-seeded code, test cases, and test case results (including any error messages). Tasked with fixing the defect-seeded code, participants then read the code, made fixes and ran tests until either the test cases passed or a time limit was reached.

Secondarily, to incorporate a partial understanding of the role of linguistic differences into our model, we also presented tasks with systematically replaced identifier names. We scope our focus to *morphemes*, the smallest meaningful unit of a linguistic expression. Following research in Psychology that multi-morpheme words can aid comprehen-

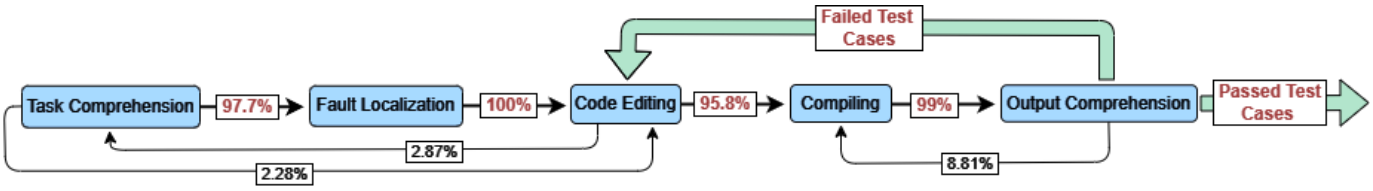


Figure 1: Proposed neurally-motivated five-stage debugging model. Participants begin in Task Comprehension and transition between stages. Edge labels indicate empirical transition frequencies. This model is intentionally direct: the novelty is not in the stages themselves but in our finding that they are neurally and behaviorally distinct.

sion in people with reading challenges [16], we investigate single- and multi-morpheme variable names (chosen from an established corpus [20]), as well as pseudoword variable names, and measure their impacts on behavioral and cognitive outcomes during debugging.

We present the following primary $(1, 2)$ and secondary $(3, 4)$ contributions and findings:

1) The first medical imaging study of the debugging process as a whole.
2) We propose a direct five-stage model of debugging (Section 2) and find that those stages are both behaviorally and neurologically distinct (Section 6.1).
3) Critically, we find no correlation between reading difficulty, morpheme complexity, or experience on behavioral outcomes (e.g., time or accuracy). This suggests that prior findings about variable names in isolation may not generalize to end-to-end debugging (Section 6.2.1).
4) Increased English reading difficulty leads to higher cognitive load in debugging. More years of experience leads to a decrease. This agrees with prior work (Section 6.3).

2 Dynamic Debugging Model

We propose a direct, staged model of end-to-end dynamic debugging. In this model, programmers transition between the following distinct stages: (1) Task Comprehension, (2) Fault Localization, (3) Code Editing, (4) Compiling, and (5) Output Comprehension. Notably, while prior work has investigated individual stages in isolation (e.g., behaviorally [21] or cognitively [9]), debugging has not been investigated neurologically as a dynamic activity. Stage dynamics are illustrated in Figure 1.

We define each proposed debugging stage in more detail. To start, *Task Comprehension* involves the programmer understanding the purpose of the program through a prose description. During *Fault Localization*, programmers navigate to problematic code and seek to identify and understand the defect. *Code Editing* is the process of changing the text of the program to repair the defect. *Compiling* includes building or running the program with the proposed fix. Finally, *Output Comprehension* involves inspecting the result of running the program (e.g., viewing test reports or stack traces, etc.).

We hypothesize that these tasks are both behaviorally and neurologically distinct, meaning the debugging stages in our model all correlate with different behavioral outcomes and neurological activity. For this reason, it is meaningful to model debugging in terms of transitions between distinct stages. We note that our model is intentionally direct: we claim no novelty in the stages, and instead focus on measuring distinctions for end-to-end debugging.

3 Background

We provide relevant background on how reading ability may impact debugging, the fNIRS neuroimaging technique, and applicable neuroscience terms.

3.1 fNIRS and Functional Neuroimaging

Functional Near-Infrared Spectroscopy (fNIRS) is a non-invasive neuroimaging technique that captures brain activity and cognitive processes. fNIRS is commonly used in psychology to study brain regions and cognitive tasks [22]–[25], including *cognitive load* [26]–[28]. Cognitive load refers to the mental effort required to perform a particular task or activity, encompassing the demands placed on working memory, attention, and overall cognitive processing. fNIRS is also increasingly used in software engineering to study correlations between brain activity and specific programming tasks (see Section 9).

fNIRS uses light to capture brain activity via source and detector nodes on a cap (see Figure 2). fNIRS measures changes in neural blood flow, called the *hemodynamic response*. Specifically, oxygenated (e.g., oxygen-rich) and deoxygenated (e.g., oxygen-poor) hemoglobin reflect light differently. When a brain region is active, it needs more oxygenated blood to fuel its metabolic demands. fNIRS captures changes in this response in real-time on *channels* between nearby source and detector pairs. The ratio of oxygenated and deoxygenated blood is the *BOLD* (blood oxygen level dependent) signal. Because of the way blood is delivered to active neural tissue, the BOLD signal and fNIRS are more accurate for the first 30 seconds of an activity [29], [30]. The BOLD signal thus places a constraint on experimental design that we return to in Section 4.2.

In contrast to other neuroimaging approaches, fNIRS is particularly appealing for studying interactive debugging because it supports ecologically-valid study designs [8]. fNIRS is non-invasive (cf. electrocorticography) and allows participants to respond to stimuli on a standard computer while sitting at a desk (cf. fMRI). Additionally, the technique does not require ionizing radiation—which can cause harm to tissue—to capture brain function (cf. PET and CT). When studying interactive debugging, fNIRS has the additional advantages of relatively high spatial resolution compared to EEG and higher temporal resolution than fMRI. fNIRS can thus be used accurately determine how activity in specific brain regions changes throughout the stages of debugging.

3.2 Identifiers, Morphemes, and Reading Difficulties

Identifier naming may impact debugging efficacy, particularly in individuals with lower English reading abilities. We

investigate this through varying the structure of one textual programming feature: variable identifier names. Specifically, we investigate if the morphological structure of identifiers impacts debugging efficacy.

Morphemes are the smallest linguistic units of meaning [17] Words can have one or more morphemes. For example, words such as "reach" feature a single unit of meaning while words such as "unhappy" have two units of meaning (e.g., "un-" + "happy"). Psychology research has found that multi-morpheme words aid the prose reading ability of individuals with reading difficulties [16], [17] Notably, this includes both non-native speakers [31], and individuals with dyslexia [16], [17], [32] (i.e., a disorder that impedes specific skills such as reading [33]). We are interested in if prose-related reading inefficiencies also impact debugging at the cognitive or behavioral level. Prior investigations have found that non-word (an unpronounceable word with no meaning) identifier names induce higher cognitive loads in code comprehension [34], but did not consider morpheme structure or end-to-end debugging.

We examine whether changes in the morpheme structures of identifiers elicit detrimental responses neurologically or in end-to-end debugging outcomes for people with lower reading ability. We include four identifier conditions in our experiment: original, single-morpheme, multi-morpheme, and pseudoword variable names (Section 4.2.2). We hypothesize that morpheme-varied identifiers will illicit an effect on people with reading inefficiencies.

3.3 Neuroscience Vocabulary and Notation

Vocabulary. The human brain has two *hemispheres*, left and right (e.g., Figure 5 shows both), and four primary *lobes*: frontal, temporal, parietal, and occipital. The frontal lobe is at the front of each hemisphere, the temporal lobe is on the side, the parietal lobe is at the top, and the occipital lobe is at the back. We focus on the cerebral cortex, the outer layer of neural tissue across all lobes. This layer is often conceptually divided into 52 distinct regions known as *Broadmann Areas* (BAs) [35]. Areas are often associated with specific cognitive functions. For example, BA 41 and 42 are associated with auditory processing. Some sets of BA regions are known by a common name, such as Broca's area (BA 44 and 45), which is located on the left hemisphere and lower frontal lobe. In this paper, we focus on the union of the brain regions identified in two previous fNIRS studies of computing [8], [9], as well as those identified as relevant in participants with reading difficulties [36].

Notation. Medical imaging analyses are typically presented in terms of *contrasts* in which activity measured during one condition or activity is compared against (subtracted from) activity measured during another (written $A > B$). Activities are often contrasted with a *resting state* measurement in which the participant is neither reading nor taking actions (e.g., the participant is waiting for a run to finish).

We report these contrasts in Section 6 as statistical t-values corresponding to each fNIRS channel. A positive t-value indicates that the specified brain area was more active during activity A than activity B, while a negative t-value indicates that the specified brain area was less active during activity A than activity B. Values farther from zero represent stronger activation contrasts between the two tasks. We report only areas with significant contrast ($p < 0.01$) after correction for multiple comparisons ($q < 0.05$).

4 Research Methodology

We conducted a neuroimaging study of interactive debugging with $n = 28$ participants.[1] We design our experiment to address three research questions:

RQ1: Does each stage of our proposed dynamic debugging model correlate with specific patterns in behavioral outcomes and neural activity?

RQ2: During debugging, how do morpheme-varied identifier names correlate with specific patterns in behavioral outcomes and neural activity?

RQ3: During debugging, how do individual skills (i.e. reading ability and programming experience) correlate with patterns of neural activity?

Our experimental protocol has three parts: (1) a demographic survey and initial programming assessment, (2) a reading skill test to proxy reading ability, and (3) a series of interactive debugging problems completed with fNIRS. Participants debugged between 6 and 13 Python programs with varied identifier names in a 50-minute session while wearing an fNIRS cap (Section 3.1). We also instrumented the debugging environment to collect key programming actions (e.g., keystrokes, window switching, etc.).

This protocol lets us capture the relevant data to address our research questions in a controlled manner. By using fNIRS, we collect neural activity needed for all three questions. By instrumenting the development environment, we can correlate this activity with the phases of our dynamic debugging model (RQ1). By including controlled variations of identifier names in the buggy programs, conducting a reading skill test, and collecting demographic data, we can also correlate neural activity with morepheme-related differences (RQ2) and individuals' skills (RQ3). In the rest of this section, we discuss our experimental design (including our fNIRS setup and debugging experiment), debugging stimuli, and recruitment in more detail.

4.1 Experimental Design

Our design focuses on three main goals: collecting relevant behavioral and neurological data to model debugging cognition, capturing participant reading ability, and maximizing ecological validity. We describe each in more detail.

4.1.1 Demographic Survey and Assessments

Demographic Survey and Programming Assessment. Before the fNIRS scan, we surveyed participants' demographic data (age, occupation, years of experience, etc.). We also asked if participants were native English speakers or had been diagnosed with a reading-related condition (e.g., dyslexia). We augmented this potentially unreliable self-reporting with a validated assessment of reading efficiency in the next stage. Additionally, participants completed a short programming quiz to assess eligibility. This assessment allowed us to control for data quality and perform statistical comparisons within participants.

1. University of Michigan IRB protocol HUM00229664.

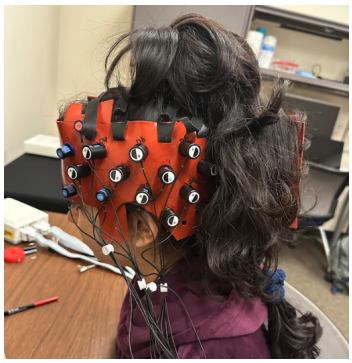


Figure 2: Image of our fNIRS cap on a participant. The cap goes around the head covering frontal and temporal regions.

Reading Ability Assessment. To capture English reading ability, we administered two *Test of Word Reading Efficiency* (TOWRE) assessments [37]. The TOWRE test is a validated measure of reading ability, widely used in psychology [38]–[40], and can distinguish English reading abilities, irrespective of reading disability diagnosis or native language.

We first use the Sight Reading Efficiency TOWRE test, in which participants read aloud a series of increasingly-complex English words as fast as possible without errors. We then use the Phonemic Decoding TOWRE test, in which participants read a list of increasingly-complex pseudowords (pronounceable words with no meaning). We used the standard scoring sheet [41] for these tests. To ensure scoring consistency, two authors independently scored recordings of participants for accuracy and then met to resolve any disagreement. We aggregate those scores into a *Total Word Reading Efficiency* (TWRE) standard score for each participant [40], [41]. Higher TWRE scores generally correlate with reading efficiency in the English language, while lower scores suggest reading difficulties or impairments.

4.1.2 Interactive Debugging

The core of our study is a 50-minute session in which participants debug a series of Python programs. For any given program, participants had at most 10 minutes to fix the bug. The debugging problems are described in Section 4.2.

Programs were displayed in Visual Studio Code (VS Code) [42], and participants could freely edit and run code. VS Code is used widely by computing students and professional developers [43], increasing the ecological validity of our experiment. Figure 3 shows our VS Code setup. Participants had access to only three files at a time per problem: a program-specific natural language task description, the buggy source code, and an output window holding testing results. After fixing the bug or reaching the 10-minute limit, participants moved to the next problem by closing the current files and opening the files related to the next task.

To minimize potential confounding variables, we did not allow participants to use a debugger, the split window editor, the internet, or revisit past problems. Prior to each study session, we showed participants a training video to acquaint them with the setup and environment restrictions.

4.1.3 Capturing Debugging Stages

To map participant neural and behavioral activity to a stage in our debugging model, we define key programmer behaviors that indicate the start and end of each non-overlapping phase based on the model description in Section 2. We define the beginning of the Task Comprehension (TC) stage to be when the participant first views the prose problem description window. The Fault Localization (FL) stage begins (and the TC stage ends) when participants view the problem code window. The Code Editing (CE) stage begins (and the previous stage ends) when participants press a non-navigational key (e.g., not an arrow key) while in the problem code window. The Compiling stage begins (and the previous stage ends) when participants instruct VS Code to run the program (either through the run button, menu option, or keyboard shortcut). The Output Comprehension (OC) stage begins (and the previous stage ends) when participants view the terminal output window.

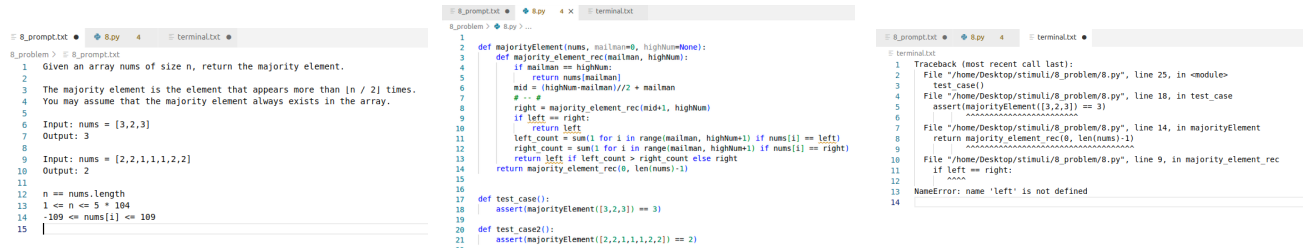
If the participant produces a successful fix (i.e., passes all tests), the current problem session ends and the cycle restarts with the next problem. However, if a fix is not successful, the participant returns to the CE stage and continues until succeeding or reaching a time limit.

We used a custom VS Code extension and build-process interception to capture relevant data such as the active file window, keystrokes, and the outcomes of program runs in real time. To ensure precise tracking of the active window (e.g., problem description vs. source code), participants were only able to view one of the three files at a time. For example, this restriction uniquely determines when a participant exits the TC phase and enters FL. In a small number of instances participants engaged in not-directly-relevant activities, such as closing multiple windows at the same time or waiting for the problem description to load. These other activities are not included in our analysis.

4.1.4 fNIRS Device, Scan and Cap Design

fNIRS Device and Scans. We measured neural activity using a 16 × 16 NIRSport 2 system. This system contains two laser diodes at 760 nm and 830 nm with fiber optic cables to transmit light between the instrument and a sensor probe on the participant's head. This system is well suited for studying debugging, capturing data of similar or higher quality than that in other fNIRS studies of programming (cf. [24]). In addition, this system supports 38 light detection channels, allowing us to capture activation from more of the brain than in many published fNIRS studies (cf. [44], [45]). Signals were sampled at 50 Hz and then resampled to 2 Hz for analysis, a common best practice [46].

fNIRS Cap Design. fNIRS experiments must select brain regions to measure. As we are interested in language-related correlates of debugging for this study, we prioritized capturing data from language-associated regions (e.g., left-hemisphere temporal lobe, Broca's region, etc.). We also covered regions identified in previous neuroimaging studies of program comprehension, data structure manipulation, and prose reading [8], [13], [36]. Our cap is a symmetrical band around the head that captures bilateral activation from lateral parts of the frontal, temporal, and parietal lobes, ultimately capturing 20 Broadmann areas (see Section 3.3). Figure 2 has a side view of the cap. We constructed four caps to accommodate different head sizes (circumference: 56 cm,



(a) Task Description (b) Program Code (c) Terminal Output Window

Figure 3: Debugging setup. Participants navigated through three windows while debugging: (a) problem description, (b) program source code, and (c) terminal window displaying testing output.

58 cm 60 cm, 62 cm) based on the international 10—20 system [47]–[49]. To fit the fNIRS cap to each participant [47], we aligned the cap's center with the 10—20 point fPZ [48] (above the bridge of the nose). Beyond our choices of which brain regions to cover, our cap construction and fitting are standard and we claim no novelty in cap design details.

4.2 Interactive Debugging Stimuli

When constructing our debugging task stimuli, we had four goals: (1) Participants should be assessed on familiar computer science topics, (2) Time taken to complete stimuli should be appropriate for fNIRS, (3) Defect seeding should be experimentally controlled, and (4) Morphemes should be experimentally varied across multi-morpheme, single-morpheme, and pseudoword (following psychology research, see Section 3.2). Because we are studying debugging and identifier morphology in the same experiment, the defect seeding and morpheme constraints were intertwined.

4.2.1 Adapting Interview-style LeetCode Problems

We wanted our stimuli to incorporate core concepts with real-world applicability, be answered within an appropriate time frame for fNIRS, and have adequate experimental control. To do so, we adapted programming stimuli from LeetCode [2], a widely-recognized online platform that provides a repository of programming problems commonly encountered in job interviews. It offers a collection of problems and solutions categorized according to difficulty levels and topics. We chose to use LeetCode as a source of stimuli due to the availability of program solutions, its large database of problems organized by difficulty, and its emphasis on fundamental programming concepts.

To keep tasks from being too complex and to maintain consistency across stimuli, we only considered LeetCode problems with Python solution lengths from 9 to 14 lines of code, plus additional lines for tests (see Figure 3b). We further restricted attention to problems admitting both controlled variation to a relevant identifier's morphological structure (Section 4.2.2) and defect seeding involving that same identifier (Section 4.2.3).

4.2.2 Selecting Identifiers to Vary Morphologically

We desire debugging scenarios in which a single relevant variable's morphological structure can be experimentally controlled. At a high level, we favor changing an identifier in a way that both follows established research in psychology and also maintain relevance to the code and defect (e.g., it appears frequently, is involved in fixing the bug, etc.).

To investigate the impact of the morphological structure on participant debugging outcomes, we consider four categories of morphemes: original, single-morpheme, multi-morpheme, and pseudoword. Original words are the unchanged LeetCode-provided identifiers, such as "target" or "total". Single-morpheme words feature a single unit of meaning, such as "reach" or "color". Multi-morpheme words involve two units of meaning, such as "replay" ("re-" + "play") or "unhappy" ("un-" + "happy"). Pseudowords are devoid of inherent meaning but are pronounceable, such as "buned" or "binsping". Pseudowords are serve a similar function as nonwords (such as those used by Siegmund *et al.* [34]), but lend the ability to contain morphemes while still being meaningless, thus acting as a stronger control.

In consultation with psychology researchers who study morphological processing, we used a validated list of morphemes from the established corpus of Marks *et al.* [20]. Their corpus contains multiple pairs of words within ± 1 length of each other such as "reuse" and "reach" (of length five), "flavor" and "visitor" (of length six and seven), etc. This allowed for controlled variation in which one morpheme is retained across the single-morpheme, multi-morpheme and pseudoword cases. For example, the single-morpheme category word "speed", the multi-morpheme category word "tried", and the pseudoword "buned" all feature the morpheme "-ed" (but only in the multi-morpheme case does it carry meaningful semantic significance). Such shared morphemes are important because neurological differences are especially notable for derivational morphology involving analytically-demanding and semantically-abstract units (e.g., "-ed", "un-", "-ly", etc.) [20].

For each word length, we filtered applicable LeetCode solutions to those containing an identifier within ± 1 of that length that also appears in close to 50% of the solution lines. In our final dataset, the average percentage of lines featuring the selected identifier was 47% (minimum 36%, maximum 64%). Once found, we change the identifier uniformly in both the code solution and also the problem description by replacing it with the single-morpheme, multi-morpheme and pseudoword elements from the Marks *et al.* corpus. For each defect scenario, a participant is shown one randomly-assigned variant from one of the four categories (original, single-morpheme, multi-morpheme, and pseudoword: see

2. https://leetcode.com/

```
def singleNumber(nums):
    lstDup = []
    for i in nums:
        # --- #
            lstDup.append(i)
        else:
            lstDup.remove(i)
    return lstDup.pop()
```

(a) Original variable name

```
def singleNumber(nums):
    brunly = []
    for i in nums:
        # --- #
            brunly.append(i)
        else:
            brunly.remove(i)
    return brunly.pop()
```

(b) Pseudoword

```
def singleNumber(nums):
    family = []
    for i in nums:
        # --- #
            family.append(i)
        else:
            family.remove(i)
    return family.pop()
```

(c) Single-morpheme

```
def singleNumber(nums):
    mostly = []
    for i in nums:
        # --- #
            mostly.append(i)
        else:
            mostly.remove(i)
    return mostly.pop()
```

(d) Multi-morpheme

Figure 4: Example stimuli with experimentally-controlled variations corresponding to identifier morphology.

Figure 4). A participant is never shown the same defect scenario twice. In total, participants were assigned 13 problems across all four identifier conditions (i.e., 3–4 problems in each). However, due to problem order randomization and timing constraints, not all participant saw all conditions.

To observe the impact of morphemes on the interactive debugging process, we also ensure that the defect (see Section 4.2.3) contains the identifier: participants must engage directly with that identifier name to fix the bug.

4.2.3 *Defect Seeding*

LeetCode solutions do not contain defects. Because we focus on studying participant behavior while debugging, we require consistent defects across problems (precluding approaches in which participants write code from scratch, since that code may or may not contain defects and two participants may or may not create defects of similar complexity). For consistency of analysis and to admit comparison to prior work [9], we *seeded* each defect by deleting a program line. For each stimulus, we removed one line of code that both contained an identifier suitable for morphological mutation and was also part of a control or data-flow dependency (e.g., an assignment or conditional statement). This ensured that participants interacted with the morphological mutation, and resulted in varied types of errors and non-trivial fixes across the stimuli. Each deletion was indicated with three consecutive dashes (see Figure 4).

While numerous operators are available for defect seeding [50], we used line-deletion defect seeding. This method is easy to understand and produces a number of different errors. Figure 4(c) produces an indentation error, but other errors such as "UnboundLocalError: cannot access local variable 'total' where it is not associated with a value," "AssertionError," and "NameError" were also produced.

At the same time, we know it is possible to fix each defect with a single line addition. This aligns with our need for participants to complete tasks quickly. Because we use the BOLD signal to measure neural activity (see Section 3.1), we desire stimuli where each debugging stage can be completed rapidly. Extended periods of engagement on the same activity can also lead to mental fatigue and reduced behavioral outcomes [51], [52].

4.3 Recruiting and Population Contextualization

We recruited using email lists, posters, and professional contacts. To reach students, we advertised at the University of Michigan via in-class presentations and online forums. To recruit participants with varying reading abilities, we collaborated with Michigan's Office of Services for Students with Disabilities to email students registered to receive accommodations for learning disabilities. This allowed us to reach students with various degrees of experience outside of computer science. We also created Chinese versions of our posters to facilitate recruiting non-native English speakers.

Participants were eligible if they were at least 18, right-handed, and had completed or were enrolled in a data structure and algorithms course. Since we studied the impact of reading ability, we gave priority to subjects who self-reported a diagnosis of dyslexia or were non-native English speakers. Potential participants were emailed a demographic survey and a link to schedule their fNIRS scan (see Section 4.1). Participants that completed the study in full were compensated $\$50$. One participant withdrew from the study during the brain scan and was compensated $\$20$.

35 participants completed the demographics form and fNIRS scan. However, 7 participants were removed from analysis due to corrupted data ($n = 4$), opting to drop out ($n = 1$), or failing to follow instructions ($n = 2$). Of the remaining 28 valid participants, 19 were undergraduates, 8 were graduate students, and 1 was a full-time software engineer. Programming experience ranged from 1 year to over 10 years with an average of 2–5 years. 6 participants identified as women, 21 as men, and 1 chose not to disclose.

Additionally, 7 participants self-identified as non-native English speakers, while 3 were diagnosed with dyslexia. This broad spectrum of language backgrounds enables us to explore English prose reading ability as a potential predictor of behavioral and neural debugging outcomes.

5 Analysis Approach

We detail our approach to analyzing neural and behavioral outcomes. We used fNIRS (Section 3.1) to measure neural activity and used standard instrumentation (e.g., recording keystrokes, timing) to measure behavioral outcomes. Critically, using the delimiters from Section 4.1.3, we associated each measurement with one stage in our debugging model.

5.1 fNIRS Data Analysis

We claim no novelty in our fNIRS analysis methodology and instead rely on established best practices from Psychology [46]. In this section, we detail the steps we took to analyze our data, including pre-processing, individual subject analysis and group-level analysis.

Preprocessing. The raw fNIRS data is in the form of light intensity values. This was converted into optical density data

(a measurement of changes in hemoglobin concentration in the brain) by calculating light absorption fluctuations associated with either oxygenated (HbO) or deoxygenated (HbR) blood. The optical density data were then converted into an HbO/HbR signal using the Modified Beer-Lambert law. We ran a general linear model (GLM) with pre-whitening and robust least squares to fit the data [53].

Individual Subject Analysis. Individual subject analysis (or first-level analysis) involves fitting general linear models (GLMs) to each subject's data independently. We specified within subject, first-level GLMs to model fNIRS optical density measurements in all the channels that were statistically related to the timing of the hemodynamic responses (as determined by the convolving timeseries of stimulus events with the canonical response function).

In fNIRS, systemic physiology and motion-induced artifacts are major sources of noise and false positives. We therefore fit our models using autoregressive-whitened robust regression [54], which controls for such confounds during parameter estimation. Then, we applied t-tests to the regression coefficients describing the task-related brain activations modeled for every participant. We additionally separated tasks by morphological structure (original, single-morpheme, multi-morpheme, pseudoword) and constructed GLMs to analyze the effect of morpheme category on neural activity for each participant.

Contrasts and Subject-Group Analysis. We computed pairwise contrasts to determine mean differences in activity between conditions on a within-participant basis. Next, we conducted a group-level analysis to summarize the first-level regression coefficients. We used a mixed effects model for the average group-level response, with individual participants treated as random effects. We used a false discovery rate threshold ($q < .05$) to control for multiple comparisons.

5.2 Behavioral Data Analysis

We used accuracy (i.e., test cases passed), window-switching frequency, and timing data to assess how identifier conditions, reading ability, and programming experience affect behavioral outcomes within dynamic debugging. Specifically, we used multi-level regression models (i.e., mixed-effects modeling) to examine relationships between our response variables (data) and independent variables (conditions and skills). We claim no novelty in the analysis methods employed themselves, and instead follow established best practices. Our use of this analysis approach [55], [56] follows that of another software engineering paper evaluating programming and cognition [57].

5.2.1 Multi-Level Regression Modeling

A mixed-effects model can have independent variables whose effects are systematic (fixed effects), heterogeneous (random effects), or both. Interactions of fixed effects allow for modeling effects that are systematic within specific groups of observations. We hypothesize morpheme-varied identifier will affect programming and neural outcomes of those with varied reading ability (based on prior work, see Section 3.2). Furthermore, random effects can model heterogeneity in the data. This is relevant for our analysis as there is a possibility that some participants may perform better or worse with varying identifier conditions.

5.2.2 Model Specifications

The dependent variables we consider are per-task accuracy (i.e., did the participant fix the bug), window-switching count, and debugging time. To ensure participants attempted multiple problems, participants moved on to the next problem after 10 minutes, even if they had not fixed the bug. In psychometrics, this is known as "right censoring", where the true time needed to solve the problem is unknown due to a time limit [58]. We mitigate censoring-related concerns by separating our time observations into: (1) per-task total response time of correctly-fixed problems, and (2) per-task total time interacting with the variable name across all observations (i.e., sum of time spent in Fault Localization, Code Editing, and Output Comprehension stages).

We apply square root transformation to response times to address skew, and fit models using maximum likelihood estimation (MLE). To find the best-fitting model for each dependant variable, we optimize Akaike Information Criterion (AIC), a widely-used model selection metric [59], [60].

During model selection, we consider effect structures for the following independent variables: morpheme-varied identifier condition, total TWRE scores (i.e., validated surrogate for reading ability), and self-reported programmer experience. Each could be a fixed or a random effect. If the best-fitting model has a fixed (systematic) effect for morpheme-varied identifier condition, TWRE scores, or experience, we explicitly verify statistical significance via a likelihood ratio "omnibus" test relative to a model without the fixed effect [56]. We pinpoint the source using post-hoc pairwise contrasts, with Benjamini-Hochberg adjustment for multiple comparisons. Alternatively, if the best-fitting model has a random (heterogeneous) identifier condition effect, we explicitly verify statistical significance using profile likelihood analysis [55] and parametric bootstrap methods [61] to find the 95% confidence bounds of the statistic.

6 Results

We present results of our investigation of the behavioral and neural correlates of interactive dynamic debugging (see section 4 for our three primary research questions).

6.1 RQ1 — Model of Dynamic Debugging

We first investigate if there is behavioral and neural distinction across the phases of our proposed debugging model (Section 2). That is, do the debugging stages in our model correlate with different behavioral outcomes and patterns of neural activity. This is important because previous approaches have considered activities such as code reading or code writing in isolation. For each model stage, we consider (1) behavioral distinction, and (2) neural distinction.

6.1.1 Behavioral Distinction of Model Stages

To test if our debugging model captures behaviorally-distinct stages (i.e., the debugging stages in our model all correlate with different behavioral outcomes), we analyzed duration and keystroke behavior during each debugging stage (Section 4.1). The average times spent in each stage were: 26.4s for Task Comprehension, 63.7s for Fault Localization, 40.7s for Code Editing, 5.9s for Compiling, and 19.2s

Brain Region	TC > Rest	FL > Rest	CE > Rest	OC > Rest	TC > CE	TC > OC	OC > CE
Frontal Cortex							
Left DLPFC (BA 46)	–	–	–	–	–	–	2.54
Right DLPFC (BA 46)	–	–	–	2.75	-2.44 – 2.69	–	2.13
Brocas Area (Left BA 44, 45)	3.41	–	–	–	2.43	2.96	2.54
IFG (Right BA 44, 45)	3.74	-2.02	-2.12 – 2.60	2.72	2.09 – 2.51	2.08	2.13
Left BA 9	–	–	–	2.05	–	–	–
Right BA 9	–	–	2.91	2.75	–	–	–
Temporal Cortex							
Wernike's Area (Left BA 22, 40)	-2.55 – 2.48	-4.25 – -2.23	–	-3.23 – -2.72	2.47 – 3.02	2.2 – 2.55	-2.64
Right BA 21	1.98	–	–	2.06	–	–	–
Left Auditory Cortex (BA 41, 42)	2.48	–	–	–	2.47 – 3.02	2.2 – 2.55	–
Right Auditory Cortex (BA 41, 42)	1.98	–	–	2.06	2.69	–	2.89
Left BA 52	2.48	–	–	–	2.47	2.2	–
Right BA 52	2.03	–	–	–	–	–	2.89
Parietal Cortex							
Left Angular Gyrus (BA 39)	–	–	2.05	–	-3.23	-2.36	-2.63
Right Angular Gyrus (BA 39)	–	–	2.91	2.75	–	–	2.62
Right BA 40	-2.35 – -4.36	-5.79 – -2.17	–	–	-3.53	–	–
Occipital Cortex							
Left BA 19	–	-2.23	–	-3.23 – -2.72	–	–	-2.63
Right BA 19	–	-3.30	–	–	–	–	–

Table 1: Statistically significant t statistics ($p < 0.05$) for each stage of our debugging model, compared to both a resting state and each other stage. "T" is "Task Comprehension", "FL" is "Fault Localization", "CE" is "Code Editing", and "OC" is "Output Comprehension". All reported values are statistically significant. Shaded cells have significant activation in the specified brain area, while non-shaded cells denote significant correlations of less activation. "BA" means Brodmann area.

for Output Comprehension. Participants spent a statistically different time in each phase ($p < 0.001$).

Keystrokes were also distinct between phases. In Code Editing, we identified 37 distinct behaviors (typing letters, deletes, keyboard shortcuts, indents, etc.). Compiling had 13 unique keyboard behaviors, and Output Comprehension had 5 unique behaviors. No keyboard activity was recorded in either Task Comprehension or Fault Localization.

Within Code Editing, the most prevalent keystrokes included typing letters (36.6% of recorded keystrokes) and deleting text (delete key, 41.0% of recorded keystrokes), totaling 4,543/12,417 and 5,098/12,417 instances respectively. In contrast, participants typed only very minimally during Compiling (226/360 keystrokes) and Output Comprehension (9/20 keystrokes). These differences between stages are strongly significant. For example, a proportion z-test reveals a significant difference in the proportion of typing letters to all keystrokes between Code Editing and Compiling ($p < 0.00001$), as well as the proportion of deletes to all keystrokes between Code Editing and Compiling ($p < 0.00001$). Participants behaved very differently across debugging stages, both in terms of time and keystrokes.

6.1.2 Neural Distinction of Model Stages

We analyzed neural activity to assess if our debugging model captures neurally-distinct stages.(i.e.,the debugging stages in our model correlate with different neurological activity). Since Compiling only involved clicking a button in our study (see Section 4), we focused attention on the other four stages: Task Comprehension, Fault Localization, Code Editing, and Output Comprehension.

To determine neural distinction between stages, for each debugging stage we identify brain activity that is statistically different between that stage and a resting state (e.g., Task Comprehension > Rest, Fault Localization > Rest, etc.), where > denotes contrast (Section 3.3). Figure 5 shows the physical locations of these distinct activation areas in the brain per debugging stage. The left half of Table 1 summarizes our results per Broadmann Area (BA) (Section 3.3). Critically, we find that our debugging stages *are* correlated to different patterns of neural activity. Each stage of our debugging model presents patterns of neural activity that vary from each other in a statistically–significant manner. We now describe how these stages differ.

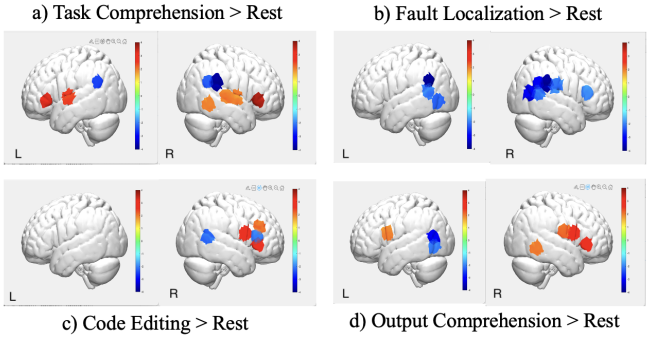


Figure 5: Contrast of debugging stages with resting state activity. 'R' is right hemisphere and 'L' left. Red indicates significant neural activation (in contrast to rest) while blue indicates significant correlation of less neural activation.

During Task Comprehension (TC) (when compared to rest), we observe substantial activation in the temporal cortex, including in Wernicke's area, right BA 21, the auditory cortex, and BA 52. Notably, this is statistically different from Code Editing (CE) ($t > 2.47$) and Output Comprehension (OC) ($t > 2.20$) which exhibit significantly less to no neural activation in the same area. TC also has more brain activation in Broca's Area. This is important as both Wernike's and Broca's area are associated with language comprehension which we define as the main activity in TC (e.g., participants make sense of the task by parsing prose descriptions).

During Code Editing (CE), we find no activation in

Wernike's area, matching prior work [5]. However, there is neural activation of the angular gyrus ($t > 2.05$) which is not present in the other stages. This is a region commonly associated with spatial cognition, spatial attention, and numerical manipulation, tasks typically connected to problem solving, which we conceptualize as the main activity in debugging. This finding highlights CE as the main phase in which problem solving takes place during debugging.

Conversely, Output Comprehension (OC) has heightened activity in the frontal cortex, with an emphasis on the right hemisphere. Interestingly, this is the only stage with significant activation in the right dorsolateral prefrontal cortex (DLPFC), a region linked to working memory ($t = 2.75$). We hypothesize that during OC, participants may be mentally stepping through the code (i.e., visualizing what step in the code caused the error), thus requiring more use of working memory. However, more investigation is needed on the neural aspects of OC for more concrete conclusions.

Intriguingly, during Fault Localization (FL) we observe no significant neural activation compared to resting state cognition. We have two hypotheses. The first is that our single-missing-line fault localization task may be easier than more general fault localization, considering participants are told where the fault is located (Section 4.2.3). The second is that FL was the longest-duration stage (64 seconds vs. 41 for CE), and it may thus represent the "default" debugging activity, with other specialized stages showing specific acute activation (e.g., OC driving activation in regions associated with working memory as participants compare the visible output to their remembered expectations). While fault localization has been investigated using eye tracking [62] or behavioral outcomes [63], [64], it has not been investigated alone using neuroimaging. A more thorough investigation of Fault Localization at the neurological level is merited.

In summary, the distinctions in behavioral activity and neural activation patterns observed during Task Comprehension, Fault Localization, Code Editing, and Output Comprehension all differ from each other in a statistically-significant manner. This provides a robust neurological foundation for our proposed end-to-end debugging model.

The stages of our debugging model are behaviorally and neurally distinct. Task Comprehension is more active in temporal regions associated with language and memory, compared to programming-intensive stages ($p < 0.01$, $q < 0.05$). Code Editing has more activation in the parietal cortex ($p < 0.004$, $q < 0.05$), which is associated reading and writing. Output Comprehension involves the DLPFC, associated with memory retrieval and working memory ($p < 0.04$, $q < 0.05$).

6.2 RQ2 — Morpheme-Varied Identifiers

We assess if identifier variations correlate with behavioral outcomes or neural activity (Section 4.2.2). This is relevant since identifiers with varied morphemes may impact programmers with reading difficulties (Section 3.2).

6.2.1 Behavioral Distinction of Morpheme-varied Identifiers

We investigate the effect of morpheme-varied identifiers on debugging behavior outcomes during dynamic debugging. Specifically, we are interested in assessing if there is a identifier condition effect on participants with reading difficulties (Section 5.2.2). We use TWRE scores as a validated measure for reading difficulty. As described in Section 4.1.1, the TWRE score is calculated as the sum of the Sight Reading Efficiency and Phonemic Decoding TOWRE tests. Our observed TWRE scores ranged from 150 to 223, with an average score of 189.9 (in the average score range for the TWRE [65]). Lower Reading Efficiency scores signify increased challenges in reading English prose. This question is directly relevant to issues of accessibility within the field, such as recruitment of non-native English speakers.

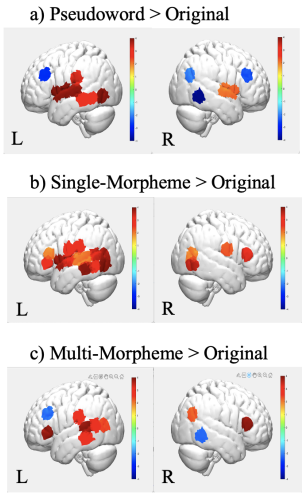


Figure 6: Contrast of neural activity during Code Editing between identifier types (single-morpheme, multi-morpheme and pseudoword) vs. original identifiers. 'R' indicates right hemisphere and 'L' left. Red spots indicate significant positive neural activation while blue spots indicate significant correlation of less neural activation when the condition is compared to the control (i.e., original).

Overall, we find no statistically-significant differences in behavioral outcomes as a function of reading ability ($p > 0.09$). That is, we find no identifier condition for which accuracy, response times, or window-switching frequency are statistically significantly different within TWRE scores or programmer experience. While absence of evidence is not evidence of absence, this finding is interesting. For example, the participants with dyslexia in our study, or in the bottom category of TWRE scores, did no worse than other participants in our study. Previous studies have suggested that individuals with reading difficulties may face challenges with morpheme-complicated prose words in the context of reading comprehension. Our findings give confidence that participants with reading difficulties, as measured by TWRE scores, may not experience negative impacts on their *end-to-end* debugging speed or accuracy. This has implications for skills assessments and their use in hiring.

We find no statistically-significant differences in speed or accuracy for end-to-end debugging as a function of Total Word Reading Efficiency score ($p > 0.09$). Over the entire debugging process, participants with reading challenges did no worse than those without.

6.2.2 Neural Distinction of Morpheme-Varied Identifiers

We investigated changes in neural activity based on identifier name morphology (Section 4.2.2). We focused on brain region activity during the Code Editing stage, as this is when participants primarily engaged with the variable identifier (Section 4.2.3). Figure 6.1.2 shows significant differ-

ences in neural activity of each treatment (single-morpheme, multi-morpheme, pseudoword) compared to original.

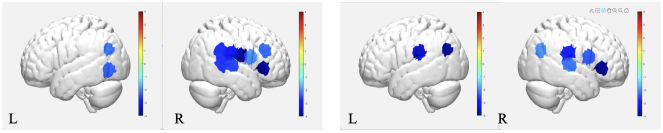
All three treatment conditions have significant activation. This suggests that all three treatments induce more cognitive load than the original identifier. All conditions exhibit significant neural activation concentrated on BA brain regions associated with Wernike and Broca's areas when compared to the original condition (t-value = 2.71 – 5.75). These regions are commonly known to be related to language. These results can be viewed as a replication of prior studies, such as those of Siegmund *et al.*, that investigated code comprehension and found that less meaningful identifiers lead to increased cognitive load [34].

We believe this is a very interesting result. The treatment conditions all induce more neural activation than the original condition. However, we found that the contrast between single-morpheme and original was more significant than the other two pairs (i.e., pseudoword > original and multi-morpheme > original). This is visualized in Figure 6.1.2. This implies that single-morpheme words induce a more uniform cognitive load than pseudowords or multi-morpheme words. This is somewhat surprising, because one might intuitively guess that single-morpheme words (like "family" or "color") would be easier to reason about than pseudowords (like "brunly" or "binsping").

We speculate that this is because the single-morpheme condition replaces an identifier name with another that has *some* atomic meaning, but that also means the *wrong* thing in context, while pseudowords can be viewed as abstract labels and multi-morpheme identifiers can still be broken down. Informally, for example, in Figure 4, if the original identifier is "lstDup", replacing it with "family" can be viewed as wrong or misleading (the variable does not hold elements with a family relation), while replacing it with "brunly", which has no meaning, is harder to reason about but is not actively misleading. In this light, our results replicate and extend those of Fakhoury *et al.*, who studied variable name "antipatterns" and found negative effects [62], [66].

We view this as an important and novel result. Prior studies have focused on meaningless identifier naming [34] or contrasted standard words to other tasks (such as finding syntax errors [13]). Our findings align with, and provide strong evidence for, hypotheses related to identifier semantics from prior work (e.g., [34, Sec. 3.1] or [62], [66]). For example, an identifier such as "bubbleSort" gives hints about purpose and sets expectations. A misleading or antipattern single-morpheme identifier gives the wrong hint or expectation. This strongly motivates further investigation of morphology, which was not explicitly considered in previous CS studies, to tease apart differences between single-morpheme and multi-morpheme conditions.

All three conditions (pseudoword, single-morpheme, multi-morpheme) show increased neural activity compared to original variables in language regions. Notably, we find that single-morpheme replacements (like "family" or "color") show a unique pattern of neural activity. Our preliminary hypothesis is that our findings align with the notions of semantic cues, beacons and antipatterns in code comprehension.



a) Years of Experience b) Reading Efficiency (TWRE)

Figure 7: Correlative neural activity for participants with high experience or TWRE scores. 'R' is right hemisphere and 'L' left. Blue spots indicate significantly less brain activation.

6.3 RQ3 — Neural Impact of Skills

We also examine the neural differences between participants with various years of experience and reading ability. Participants completed a demographic survey that included self-reported programming experience (Section 4.1.1).

As years of experience and TWRE scores increase, there is less neural activation (see Figure 7). Brain regions less associated with significant activation as experience increased and reading ability increased are concentrated on the right hemisphere in left BA 39, right BA 22, right BA 44 and 45, and right BA 41 and 42, with t value -2.07 – -5.35. These results align with classic neuropsychology findings that the brains of experts show lower metabolic activity (i.e., less energy required) to produce the same amount of neural activity [67]. They also align with prior findings (e.g., Floyd *et al.* [68]) that increased experience with computing results in distinct patterns of neural activity.

Interestingly, participants with more years of experience exhibit less widespread neural activation than those with higher TWRE scores, suggesting that computing expertise may have a more pronounced impact in reducing cognitive load during debugging than does reading efficiency. We view this as a positive result for broadening participation in computing, since it suggests that programmers with lower incoming reading efficiencies (such as non-native speakers or developers with dyslexia) can still obtain increased efficiency and reduced cognitive load. This is best interpreted in light of our results in RQ2 and RQ3 that participants with lower TWRE scores showed no statistically-significant differences in end-to-end debugging outcomes.

Years of experience and higher TWRE scores (i.e., reading efficiency in prose English) statistically-significantly decrease cognitive load in novice programmers ($p < 0.001$, $t = -5.35$ – -2.38 and $p < 0.001$, $t = -4.46$ – -2.073, respectively). These results align with prior medical imaging findings on expertise and suggest that both computing and natural language expertise can decrease cognitive load during debugging.

7 Threats to Validity

We consider several relevant threats to validity.

Construct Validity. Our identifier conditions may not reflect the meaningful names emphasized in conventional software development. We prioritized using morphological-focused words as identifier names. We used a validated list of morphemes from the established corpus of Marks *et al.* [20].

In addition, the programs used in our stimuli are smaller than typical industrial software and may not generalize to

industrial practice. To address this risk, we chose to use interview questions used in software developer hiring, which are still indicative of a programming task (job-seeking).

Internal Validity. Assessments of "reading inefficiency" and "experience" may introduce threats to internal validity, as self-reporting is not always perfectly reliable. We mitigated the risk of inaccurate self-reported reading ability by administering a validated reading ability assessment to participants (see Section 4.1.1). For self-reported experience levels, we acknowledge the difficulty in verifying accuracy. However, our findings related to neural activity patterns based on self-reported computing experience align with prior work [68] that examined self-reported courses and grades. This consistency increases our confidence in the reliability of self-reported experience levels for this study.

Ecological Validity. Participants could only view one window in the editor (see Section 4.1.3), unlike real-world practices. This restriction added confidence that brain activation was due to a specific task and not from viewing multiple windows simultaneously. Participants were also allowed to switch back and forth between windows as frequently as they wished, which is closer to real-world practice. We hope future research will explore designs incorporating split editor windows to further enhance ecological validity.

Generalizability of Results. Our defect seeding method may not encompass all types of software bugs, affecting generalizability. We aimed to enhance generalizibility by designing stimuli that generated a wide range of error messages. In addition, we vary the complexity of the problems to encompass wider skills required by software engineers. This approach allowed us to present participants with diverse debugging scenarios, thus broadening the applicability of our findings. A detailed explanation of our defect seeding process is discussed in Section 4.2.3. Additionally, since only one participant was a professional developer (Section 4.3), our results may be most relevant to novices.

Debugging Model Limitations. Our cognitive model for debugging focuses on aspects of task comprehension, fault localization of a single bug, code editing, and output comprehension. It does not consider factors such as programming languages, development environments, or other biases, which may significantly impact the results. We hope that future work explores these additional factors.

fNIRS Cap Design. There may have been significant differences in brain regions not covered by our fNIRS cap. To mitigate this, we adapted a validated cap design used by Huang et al. for a software engineering study [8]. We also considered areas identified in other medical imaging studies (e.g. with fMRI [13], [34]). Our modern fNIRS system, with its large number of channels, allowed us to consider the union of all relevant brain regions reported in related work.

8 Discussion

We discuss how our findings relate to prior work, as well implications for developers with reading impairments.

8.1 Alignment with Prior Work

We found statistically-significant neural distinctions across the phases of our debugging model, aligning with existing research. First, our Task Comprehension stage (where participants engage with English program descriptions) aligns with prior work on the neural correlates of prose reading [5]. Similarly, neural activity observed during Code Editing corroborates prior findings from Kruger *et al.* [9]. Together, these strengthen the growing understanding that neural processes during reading and code editing are distinct.

Our Code Editing stage, where participants edit buggy code with the goal of finding a fix, shows neural activation in areas often associated with spatial cognition. Prior work in software engineering has found that data structure manipulation is also heavily linked to spatial cognition. These findings led to studies [6], [69], which used training regimes like mentally rotating objects to see if the training transfers to other skills, such as data structure manipulation. We recommend similar studies focused on the skill of editing.

Our Output Comprehension stage, in which participants review testing output, presents an intriguing intersection of neural behaviors reminiscent of both Task Comprehension and Code Editing. We speculate that this is because Output Comprehension involves both code-related interpretation (e.g., stack traces) and reasoning about the (prose) problem description (i.e., what the correct output should be).

Our results extend beyond understanding neural distinctions, holding potential implications for CS pedagogy. For example, prior work has found that technical prose reading training can improve CS outcomes, especially in activities such as tracing through code [5], [6]. Our results directly suggest such training may also benefit students or programmers during testing and test output comprehension.

8.2 Reading Ability and Variable Naming

While we did observe significant differences in patterns of neural activity as a function of identifier morphology or TWRE scores, we did not find that to carry through to end-to-end behavioral outcomes such as debugging speed or accuracy. This observation holds true across varying levels programming experience. We view this as an important distinction between our work and prior work, and one that stems from our consideration of end-to-end, integrated debugging. Prior studies reported increased cognitive load (and differing neural activity) with changes to variable names, a finding we replicate, but did not delve as deeply into end-to-end accuracy or speed. While cognitive load findings inform the design of better support for programmers, we ultimately find that participants with less English reading efficiency *perform just as well overall* as those with more reading efficiency. This has the potential to partially dispel harmful prejudices about hiring or collaborating with non-native speakers or those with reading challenges.

9 Related Work

We overview prior research on the study of software engineering using medical imaging, the consequences of inadequate identifier naming, and programmers with dyslexia.

9.1 Software Engineering and Neuroimaging

In recent years, interest in using neuroimaging as a method to understand various software engineering tasks at the

cognitive level has increased. Studies have investigated the neural correlates of various software engineering tasks such as code comprehension, code reading, and code writing. Siegmund *et al.* investigated the neurological patterns associated with programming comprehension, specifically focusing on short snippets of Python code that included embedded syntax errors to observe corresponding neural responses [13]. Siegmund *et al.* further investigated the role of semantic cues in code comprehension [34]. Kreguer *et al.* investigated code writing and prose writing, finding these tasks to be neurally distinct [9], with subsequent work also finding differences at the connectivity level [70].

Neuroimaging studies have also looked into the relationship between data structure manipulation and mental rotation. Yu *et al.* compared data structure manipulation to mental rotation using both fMRI and fNIRS [8], finding significant overlap. Endres *et al.* investigated the impact of technical reading training and spatial skills training on CS1 student performance, including using fNIRS [5], [6].

Researchers have also been interested in understanding the neural correlates of bug detection in isolation using fMRI. Duraes *et al.* examined the neural correlates associated with code comprehension before bug detection and at the time the programmer found a bug [12]. Their findings on code comprehension further supported previous evidence that code comprehension involves brain regions associated with language processing and mathematics. Castelhano *et al.* found that the insula, a region deep within the brain and associated with emotional processing and interception, plays a distinct role in bug monitoring and bug detection [11].

Alternatively, our study focuses on the debugging process as a whole, including comprehension, code writing, code reading, and bug detection. The stimuli used in many of prior studies were smaller and less reflective of real-world programming due to the constraints of fMRI. Our participants use VS Code, mirroring a real-world scenario and enhancing the applicability of our findings. In general, our per-stage findings agree with prior studies that focused on each stage alone. However, these differences in patterns of neural activity and cognitive load, while statistically significant, did *not* result in end-to-end speed or accuracy differences — an observation that requires considering debugging as a whole, rather than on stages in isolation.

9.2 Identifier Naming Conventions

Researchers have investigated what makes good software engineering identifiers. Short identifiers have been compared to long identifiers [71], [72] and abbreviated identifiers have been compared to full-length identifiers [73]–[75].

Different identifier styles have been compared, particularly camelCase vs. snake_case [76]–[78]. Al Madi and Zang looked at the effect of lexically-similar names on debugging [79], [80]. By contrast, we consider morphologically-different identifiers. In psychology, prior work has highlighted the role of morphological awareness in the development of reading skills and how morphological skills can uniquely predict reading efficiency [81], [82].

Fakhoury *et al.* used fNIRS and eye tracking to study the effects of linguistic antipatterns in identifier names on neural activity for software comprehension tasks in [62] and for debugging in [66]. Identifiers which follow linguistic antipatterns are perceived negatively by developers. The identifiers usually have a misleading name (e.g., an integer variable named "numbers"). Our findings related to single-morpheme words (but semantically incorrect) identifiers strongly agree with those of Fakhoury *et al.*.

9.3 Empirical Analysis of Debugging

Two early models of debugging are in Katz *et al.* [83] and Vessey [84]. Both have similar basic components (e.g., test, locate, and repair). Gilmore presented a new model [85], integrating the debugging into a larger model of normal programming activities. Many more recent models of debugging also consider debugging as part of the overall development process [86], [87]. We do not consider debugging as part of an overall development process, instead focusing on the activities within debugging itself. To the best of our knowledge, prior models do not include neurological evidence that their stages are cognitively distinct. Ahrens *et al.* used eye tracking, but not medical imaging, to investigate debugging [88]. Their study focused on attention, rather than modeling debugging stages. By contrast, we consider a direct model of debugging and provide evidence that its stages are both behaviorally and neurologically distinct.

10 Conclusion

We propose a simple debugging model in which programmers transition between Task Comprehension, Fault Localization, Code Editing, Compiling, and Output Comprehension. We conduct a human study of $n = 28$ participants using fNIRS and standard coding measurements (e.g., time taken, tests passed, etc.). We also consider the role of identifier morphology, a concept used in psychology, to assess participants with dyslexia and other reading inefficiencies. We considered the four morpheme conditions of original, single-morpheme, multi-morpheme, and pseudoword.

We find that our proposed stages of debugging correlate with distinct neural activity and behavioral outcomes. To the best of our knowledge, this is the *first neurally-justified cognitive model of debugging as a whole*. Second, we replicate prior neuroimaging results about individual stages of debugging, including observing that misleading identifiers induce a higher cognitive load, while increasing years of experience is associated with a lower cognitive load. Third, we find *no speed or accuracy differences* in end-to-end debugging as a function of English reading efficiency. We see this as an important result that may help dispel some prejudices related to broader participation in computing, such as non-native speakers or participants with reading inefficiencies.

11 Acknowledgments

This work was partially supported by NSF grant #2211749. We thank Ioulia Kovelman for generously allowing us to use her fNIRS device. We thank her students, James and Rachel, for their expertise on reading ability, morphemes, and brain activity. Our thanks extend to Chi-lin for his assistance in shaping the experimental design. We thank Frank for his support in the analysis of the fNIRS data and in generating results. Lastly, we also thank Emma for her valuable insights into multi-level regression.

REFERENCES

[1] D. H. O'Dell, "The debugging mindset: Understanding the psychology of learning strategies leads to effective problem-solving skills." *Queue*, vol. 15, no. 1, pp. 71–90, 2017.

[2] C. L. Goues, M. Pradel, and A. Roychoudhury, "Automated program repair," *Commun. ACM*, vol. 38, no. 04, p. 56–65, 2019.

[3] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[4] M. Endres, A. Brechmann, B. Sharif, W. Weimer, and J. Siegmund, "Foundations for a new perspective of understanding programming (#22402)," *Dagstuhl Reports*, vol. 12, no. 10, pp. 61–83, 2022.

[5] M. Endres, Z. Karas, X. Hu, I. Kovelman, and W. Weimer, "Relating reading, visualization, and coding for new programmers: A neuroimaging study," in *International Conference on Software Engineering*, 2021, pp. 600–612.

[6] M. Endres, M. Fansher, P. Shah, and W. Weimer, "To read or to rotate? comparing the effects of technical reading training and spatial skills training on novice programming ability," in *Foundations of Software Engineering*. ACM, 2021, pp. 754–766.

[7] D. H. Uttal, N. G. Meadow, E. Tipton, L. L. Hand, A. R. Alden, C. Warren, and N. S. Newcombe, "The malleability of spatial skills: A meta-analysis of training studies." *Psychological bulletin*, vol. 139, no. 2, p. 352, 2013.

[8] 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," *Proceedings of the 41st International Conference on Software Engineering*, vol. 1, p. 396–407, 2019.

[9] R. Krueger, Y. Huang, X. Liu, T. Santander, W. Weimer, and K. Leach, "Neurological divide: An FMRI study of prose and code writing," in *International Conference on Software Engineering*, 2020, p. 678–690.

[10] Z. Sharafi, Y. Huang, K. Leach, and W. Weimer, "Toward an objective measure of developers' cognitive activities," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–40, 2021.

[11] J. Castelhano, I. C. 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. 623–637, 2019.

[12] J. Duraes, H. Madeira, J. Castelhano, C. Duarte, and M. C. Branco, "Wap: Understanding the brain at software debugging," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. Piscataway, NJ, USA: IEEE, 2016, pp. 87–92.

[13] 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 *International Conference on Software Engineering*, 2014, pp. 378–389.

[14] D. Posnett, V. Filkov, and P. Devanbu, "Ecological inference in empirical software engineering," in *Automated Software Engineering*, 2011, pp. 362–371.

[15] W. Aspray, F. Mayadas, and M. Y. Vardi, "Globalization and offshoring of software," in *The Innovation Imperative*. Edward Elgar Publishing, 2009, pp. 171–173.

[16] J. M. Law, A. Veispak, J. Vanderauwera, and P. Ghesquière, "Morphological awareness and visual processing of derivational morphology in high-functioning adults with dyslexia: An avenue to compensation?" *Applied Psycholinguistics*, vol. 39, no. 3, pp. 483–506, 2018.

[17] J. M. Law, J. Wouters, and P. Ghesquière, "Morphological awareness and its role in compensation in adults with dyslexia," *Dyslexia*, vol. 21, no. 3, pp. 254–272, 2015.

[18] M. Behroozi, C. Parnin, and T. Barik, "Hiring is broken: What do developers say about technical interviews?" in *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2019, pp. 1–9.

[19] J. Harper, "Interview insight: How to get the job," in *A Software Engineer's Guide to Seniority: A Guide to Technical Leadership*. Berkley, CA: Apress, 2022, pp. 19–28.

[20] R. A. Marks, R. L. Eggleston, X. Sun, C.-L. Yu, K. Zhang, N. Nickerson, X.-S. Hu, and I. Kovelman, "The neurocognitive basis of morphological processing in typical and impaired readers," *Annals of Dyslexia*, vol. 72, no. 2, pp. 361–383, 2022.

[21] Z. P. Fry, B. Landau, and W. Weimer, "A human study of patch maintainability," in *ISSTA*. ACM, 2012, pp. 177–187.

[22] M. M. Arredondo, "Shining a light on cultural neuroscience: Recommendations on the use of fnirs to study how sociocultural contexts shape the brain." *Cultural Diversity and Ethnic Minority Psychology*, vol. 29, no. 1, p. 106, 2023.

[23] R. Adorni, A. Gatti, A. Brugnera, K. Sakatani, and A. Compare, "Could fnirs promote neuroscience approach in clinical psychology?" p. 456, 2016.

[24] X. Hu, C. Zhuang, F. Wang, Y.-J. Liu, C.-H. Im, and D. Zhang, "fnirs evidence for recognizably different positive emotions," *Frontiers in human neuroscience*, vol. 13, p. 120, 2019.

[25] M. Soltanlou, M. A. Sitnikova, H.-C. Nuerk, and T. Dresler, "Applications of functional near-infrared spectroscopy (fnirs) in studying cognitive development: The case of mathematics and language," *Frontiers in psychology*, vol. 9, p. 277, 2018.

[26] S. Karmakar, S. Kamilya, P. Dey, P. K. Guhathakurta, M. Dalui, T. K. Bera, S. Halder, C. Koley, T. Pal, and A. Basu, "Real time detection of cognitive load using fnirs: A deep learning approach," *Biomedical Signal Processing and Control*, vol. 80, p. 104227, 2023.

[27] N. Taylor, M. Wyres, M. Bollard, and R. Kneafsey, "Use of functional near-infrared spectroscopy to evaluate cognitive change when using healthcare simulation tools," *BMJ Simulation & Technology Enhanced Learning*, vol. 6, no. 6, p. 360, 2020.

[28] M. A. Khan, H. Asadi, T. Hoang, C. P. Lim, and S. Nahavandi, "Measuring cognitive load: Leveraging fnirs and machine learning for classification of workload levels," in *International Conference on Neural Information Processing*. Singapore: Springer, 2023, pp. 313–325.

[29] M. A. Lindquist, J. Meng Loh, L. Y. Atlas, and T. D. Wager, "Modeling the hemodynamic response function in fmri: Efficiency, bias and mis-modeling," *NeuroImage*, vol. 45, no. 1, Supplement 1, pp. S187–S198, 2009, mathematics in Brain Imaging.

[30] A. Bonilauri, F. Sangiuliano Intra, G. Baselli, and F. Baglio, "Assessment of fnirs signal processing pipelines: Towards clinical applications," *Applied Sciences*, vol. 12, no. 1, p. 316, 2022.

[31] Y. Hayashi and V. Murphy, "An investigation of morphological awareness in japanese learners of english," *Language Learning Journal*, vol. 39, no. 1, pp. 105–120, 2011.

[32] M. Leikin and E. Z. Hagit, "Morphological processing in adult dyslexia," *Journal of psycholinguistic research*, vol. 35, pp. 471–490, November 2006.

[33] D. American Psychiatric Association, A. P. Association *et al.*, *Diagnostic and statistical manual of mental disorders: DSM-5*. Washington, DC, USA: American psychiatric association, 2013, vol. 5.

[34] 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 *Foundations of Software Engineering*. ESEC/FSE, 2017, pp. 140–150.

[35] L. J. Garey, "Brodmann's' localisation in the cerebral cortex'," 1999.

[36] K. Zhang, X. Sun, C.-L. Yu, R. L. Eggleston, R. A. Marks, N. Nickerson, V. C. Caruso, X.-S. Hu, T. Tardif, T.-L. Chou, J. R. Booth, and I. Kovelman, "Phonological and morphological literacy skills in english and chinese: A cross-linguistic neuroimaging comparison of Chinese-English bilingual and monolingual english children," *Hum Brain Mapp*, vol. 44, no. 13, pp. 4812–4829, Jul. 2023.

[37] J. K. Torgesen, R. K. Wagner, and C. A. Rashotte, "Test of word reading efficiency–second edition," *Pro-Ed*, 2012.

[38] M. Thambirajah, "Developmental dyslexia: clinical aspects," *Advances in psychiatric treatment*, vol. 16, no. 5, pp. 380–387, 2010.

[39] R. Akhil, A. Soori, D. Shankar, M. M. Krishnan, and B. R. Poorna, "Detecting specific learning disabilities," in *2017 International Conference on Networks & Advances in Computational Technologies (NetACT)*. Thiruvananthapuram, India: IEEE, 2017, pp. 359–363.

[40] J. M. Tarar, E. B. Meisinger, and R. H. Dickens, "Test review: Test of word reading efficiency–second edition by torgesen, jk, wagner, rk, & rashotte, ca," 2015.

[41] J. K. Torgeson, R. K. Wagner, and C. A. Rashotte, *Test Review: Test of Word Reading Efficiency (TOWRE)*. Austin, Texas: Pro-ed, 1999.

[42] Microsoft, "Visual studio code-code editing. redefined," Nov 2021. [Online]. Available: https://code.visualstudio.com

[43] Stack Overflow Team, "Stack overflow developer survey 2019," accessed on December, 2023. [Online]. Available: https://insights.stackoverflow.com/survey/2019\#development-environments-and-tools

[44] 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 *Companion to the international conference on software engineering*. New York, NY, USA: ICSE, 2014, pp. 448–451.

[45] Y. Ikutani and H. Uwano, "Brain activity measurement during program comprehension with nirs," in *15th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. NJ, USA: IEEE, 2014, pp. 1–6.

[46] M. A. Yücel, A. v. Lühmann, F. Scholkmann, J. Gervain, I. Dan, H. Ayaz, D. Boas, R. J. Cooper, J. Culver, C. E. Elwell, A. Eggebrecht, M. A. Franceschini, C. Grova, F. Homae, F. Lesage, H. Obrig, I. Tachtsidis, S. Tak, Y. Tong, A. Torricelli, H. Wabnitz, and M. Wolf, "Best practices for fNIRS publications," *Neurophotonics*, vol. 8, no. 1, p. 012101, 2021.

[47] H. H. Jasper, "Ten-twenty electrode system of the international federation," *Electroencephalogr Clin Neurophysiol*, vol. 10, pp. 371–375, 1958. [Online]. Available: https://cir.nii.ac.jp/crid/1571698600671094016

[48] U. Herwig, P. Satrapi, and C. Schönfeldt-Lecuona, "Using the international 10-20 eeg system for positioning of transcranial magnetic stimulation," *Brain topography*, vol. 16, pp. 95–99, 2003.

[49] X. Xiao, H. Zhu, W.-J. Liu, X.-T. Yu, L. Duan, Z. Li, and C.-Z. Zhu, "Semi-automatic 10/20 identification method for mri-free probe placement in transcranial brain mapping techniques," *Frontiers in neuroscience*, vol. 11, p. 4, 2017.

[50] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

[51] M. A. Boksem and M. Tops, "Mental fatigue: costs and benefits," *Brain research reviews*, vol. 59, no. 1, pp. 125–139, 2008.

[52] M. Gergelyfi, B. Jacob, E. Olivier, and A. Zénon, "Dissociation between mental fatigue and motivational state during prolonged mental activity," *Front. in behavioral neuroscience*, vol. 9, p. 176, 2015.

[53] X. Cui, S. L. Bray, and A. L. Reiss, "Functional near infrared spectroscopy (nirs) signal improvement based on negative correlation between oxygenated and deoxygenated hemoglobin dynamics," *NeuroImage*, vol. 49, pp. 3039–3046, 2010.

[54] X.-S. Hu, N. Wagley, A. T. Rioboo, A. F. DaSilva, and I. Kovelman, "Photogrammetry-based stereoscopic optode registration method for functional near-infrared spectroscopy," *Journal of Biomedical Optics*, vol. 25, no. 9, p. 095001, 2020.

[55] D. Bates, M. Mächler, B. Bolker, and S. Walker, "Fitting linear mixed-effects models using lme4," *Journal of Statistical Software*, vol. 67, no. 1, p. 1–48, 2015.

[56] J. J. Faraway, *Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models*. New York, NY, USA: CRC press, 2016.

[57] H. Ahmad, M. Endres, K. Newman, P. Santiesteban, E. Shedden, and W. Weimer, "Causal relationships and programming outcomes: A transcranial magnetic stimulation experiment," in *46th International Conference on Software Engineering*, 2024, pp. 1–13.

[58] A. J. Turkson, F. Ayiah-Mensah, and V. Nimoh, "Handling censoring and censored data in survival analysis: A standalone systematic literature review," *International Journal of Mathematics and Mathematical Sciences*, vol. 2021, p. 16, Sep 2021.

[59] J. E. Cavanaugh and A. A. Neath, "The akaike information criterion: Background, derivation, properties, application, interpretation, and refinements," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 11, no. 3, p. e1460, 2019.

[60] H. Akaike, "Information theory and an extension of the maximum likelihood principle," Budapest, Hungary, pp. 267–281, 1973.

[61] A. C. Davison and D. V. Hinkley, *Bootstrap Methods and their Application*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

[62] S. Fakhoury, D. Roy, Y. Ma, V. Arnaoudova, and O. Adesope, "Measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization," *Empirical Software Engineering*, vol. 25, pp. 2140–2178, May 2020.

[63] Z. P. Fry and W. Weimer, "A human study of fault localization accuracy," in *International Conference on Software Maintenance*. Piscataway, NJ, USA: IEEE, 2010, pp. 1–10.

[64] P. Santiesteban, Y. Huang, W. Weimer, and H. Ahmad, "Cirfix: Automated hardware repair and its real-world applications," *IEEE Transactions on Software Engineering*, vol. 49, no. 7, 2023.

[65] E. Nightingale, D. Greenberg, and L. Branum-Martin, "Selecting fluency assessments for adult learners." *Grantee Submission*, vol. 5, pp. 18–29, 2016.

[66] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *International Conference on Program Comprehension*. Piscataway, NJ, USA: IEEE, 2018, pp. 286–296.

[67] U. Debarnot, M. Sperduti, F. Di Rienzo, and A. Guillot, "Experts bodies, experts minds: How physical and mental training shape the brain," *Frontiers in Human Neuroscience*, vol. 8, p. 280, May 2014.

[68] B. Floyd, T. Santander, and W. Weimer, "Decoding the representation of code in the brain: An fmri study of code review and expertise," in *International Conference on Software Engineering*, 2017, pp. 175–186.

[69] S. Sorby, B. Casey, N. Veurink, and A. Dulaney, "The role of spatial training in improving spatial and calculus performance in engineering students," *Learning and Individual Differences*, vol. 26, pp. 20–29, 2013.

[70] Z. Karas, A. Jahn, W. Weimer, and Y. Huang, "Connecting the dots: Rethinking the relationship between code and prose writing with functional connectivity," in *Foundations of Software Engineering*, 2021, p. 767–779.

[71] J. C. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," *Empirical Software Engineering*, vol. 24, no. 1, pp. 417–443, Feb 2019.

[72] D. Binkley, D. Lawrie, S. Maex, and C. Morrell, "Identifier length and limited programmer memory," *Science of Computer Programming*, vol. 74, no. 7, pp. 430–445, 2009.

[73] G. Scanniello, M. Risi, P. Tramontana, and S. Romano, "Fixing faults in c and java source code: Abbreviated vs.full-word identifier names," *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 2, pp. 1–43, 2017.

[74] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a name? a study of identifiers," in *14th International Conference on Program Comprehension (ICPC'06)*. NJ, USA: IEEE, 2006, pp. 3–12.

[75] P. Tramontana, M. Risi, and G. Scanniello, "Studying abbreviated vs. full-word identifier names when dealing with faults: An external replication," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: Association for Computing Machinery, 2014.

[76] D. Binkley, M. Davis, D. Lawrie, and C. Morrell, "To camelcase or under_score," in *2009 IEEE 17th International Conference on Program Comprehension*. Vancouver, BC, Canada: IEEE, 2009, pp. 158–167.

[77] D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell, and B. Sharif, "The impact of identifier style on effort and comprehension," *Empirical Software Engineering*, vol. 18, no. 2, pp. 219–276, Apr 2013.

[78] B. Sharif and J. I. Maletic, "An eye tracking study on camelcase and under_score identifier styles," in *18th International Conference on Program Comprehension*. NJ, USA: IEEE, 2010, pp. 196–205.

[79] N. Al Madi and M. Zang, "Would a rose by any other name smell as sweet? examining the cost of similarity in identifier naming," in *2022 33rd Annual Workshop on the Psychology of Programming Interest*, Psychology of Programming Interest Group (PPIG). Simpson, UK: Psychology of Programming Interest Group (PPIG), 08 2022.

[80] N. Al Madi, "Namesake: A checker of lexical similarity in identifier names," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023.

[81] W. Nagy, V. W. Berninger, and R. D. Abbott, "Contributions of morphology beyond phonology to literacy outcomes of upper elementary and middle-school students." *Journal of educational psychology*, vol. 98, no. 1, p. 134, 2006.

[82] J. F. Carlisle and C. A. Stone, "Exploring the role of morphemes in word reading," *Reading research quarterly*, vol. 40, no. 4, pp. 428–449, 2005.

[83] I. R. Katz and J. R. Anderson, "Debugging: An analysis of bug-location strategies," *Hum.-Comput. Interact.*, vol. 3, no. 4, p. 351–399, dec 1987.

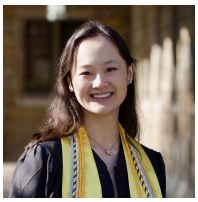
[84] I. Vessey, "Expertise in debugging computer programs: A process analysis," *International Journal of Man-Machine Studies*, vol. 23, no. 5, pp. 459–494, 1985.

[85] D. J. Gilmore, "Models of debugging," *Acta Psychologica*, vol. 78, no. 1, pp. 151–172, 1991.

[86] C. S. Corley, F. Lois, and S. Quezada, "Web usage patterns of developers," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 381–390.

[87] J. a. Lourenço and J. C. Cunha, "The pdbg process-level debugger for parallel and distributed programs," in *Symposium on Parallel and Distributed Tools*, ser. SPDT '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 154.

[88] M. Ahrens, K. Schneider, and M. Busch, "Attention in software maintenance: An eye tracking study," in *6th International Workshop on Eye Movements in Programming (EMIP)*, 2019, pp. 2–9.



Danniell Hu received a BSE in Computer Science from the University of Michigan and is currently a PhD student in the Realize lab at the University of Michigan being advised by Elizabeth Bondi–Kelly. Her research interests are broadly in the area of AI for social impact, public health, and healthcare.

Priscila Santiesteban received a BA in Computer Science and Physics from Coe College and is currently a PhD student at the University of Michigan being advised by Westley Weimer. Her research interests relate to software engineering with an emphasis on human factors and programming.

Madeline Endres received a BS in Computer Science and PhD in Computer Science and Engineering from the University of Michigan. She will be starting as an Assistant Professor of Computer Science at the University of Massachusetts Amherst in January 2025. Her main research interests lie at the intersection of Software Engineering, programming languages, and human factors, focusing on understanding and improving programmer productivity and well-being.

Westley Weimer received a BA in Computer Science and Mathematics from Cornell University, and an MS and a PhD in Computer Engineering from the University of California, Berkeley. He is currently a Professor of Computer Science at the University of Michigan. His main research interests include static and dynamic analyses to improve software quality and fix defects, as well as medical imaging and human studies of programming.