Understanding *understanding*: How do we reason about computational logic?

Ph.D. Dissertation Proposal
Hammad Ahmad (hammada@umich.edu)

Overview:

We propose to better understand user cognition for logical reasoning in software engineering using a variety of objective measures (ranging from functional to physiological to medical). We focus on using programming tools to facilitate **digital logic** design and debugging, using eye-tracking methodologies to investigate problem solving for **mathematical logic**, and using transcranial magnetic stimulation (TMS) to codify the relationship between **programming logic** and spatial reasoning.

Intellectual Merit:

This dissertation proposal argues for **objective**, **non-intrusive** measures to better understand student cognition for logical reasoning in computer science. We propose to use functional, physiological, and medical methodologies to construct **context-specific mathematical models** for logical cognition that account for student **incoming preparation** or expertise. We answer questions such as: *Can we use automated tools to help students debug defects in hardware designs? Can we use eye-tracking to better investigate how students understand proofs about algorithms? If we stimulate brain regions associated with spatial reasoning, does that impact how students reason about code?* We propose three research efforts:

(1) Providing automated debugging assistance to hardware designers. We propose to develop the first-of-its-kind automated program repair (APR) tool for hardware designs (i.e., digital logic), and to investigate its use as debugging aid for novice and expert designers.

(2) Understanding cognition for computer science formalisms. We propose to use eye-tracking to gather novel insights into how students with different levels of incoming preparation understand algorithmic proofs (i.e., mathematical logic), and what educators can do to better teach formalisms to students.

(3) Investigating causality between spatial reasoning and programming. We propose to codify the relationship between spatial reasoning and program comprehension (i.e., programming logic) using transcranial magnetic stimulation, demonstrating the first ever use of the technique for software engineering.

Broader Impact:

Our proposed research efforts put an emphasis on **mentorship** and **pedagogy**. We will actively involve undergraduate students in the proposed research agenda, with a special focus on students from groups typically underrepresented in computer science. For instance, Zachary Karas (a returning adult student) and Priscila Santiesteban (a first-generation, Hispanic pre-doctoral student) were mentored and are authors on publications associated with the research efforts for this proposal. PI Ahmad's NSF REU text was funded ($8,000 total) to fully support an additional undergraduate student for our proposed research.

Our research will delve into recommendations for educators and propose future intervention studies to investigate these suggested approaches. For instance, PI Ahmad has been approached by educators at the University of Michigan and University of Washington on using preliminary results from our research to better teach undergraduate theory and hardware courses. Our research also aims at broadening participation in computer science by formally modeling incoming preparation as an independent variable, and thus has the potential to be applied to a wider variety of student groups.

1 Introduction

Reasoning about computational logic not only constitutes a fundamental pillar of software engineering activities [27, 57, 73], but also forms a core component of undergraduate computer science curricula [1]. Indeed, many introductory computer science courses are structured around cultivating critical thinking and problem solving abilities using logical reasoning [6, 58]. While there exist several interpretations of logic (e.g., the study of correct reasoning or deduction in Philosophy [64]), in this proposal, we consider three facets of logic related to computation: *digital logic* [29] (e.g., hardware designs), *mathematical logic* [50] (e.g., proofs about algorithms), and *programming logic* [69] (e.g., coding, manipulating data structures).

Given its importance to the field of computer science (both education and industrial practice), we are interested in investigating how educators can better teach logical reasoning to undergraduate students. Recent neuroimaging studies of programmers have shown the potential to inform pedagogy and guide tool development and retraining (see Floyd *et al.* [20, Sec. II-D] for a summary). This potential was recently investigated in a study comparing the pedagogical effects of cognitive interventions guided by neuroimaging studies [17]. This study, and others [83, 84], have demonstrated that cognitive interventions (e.g., a technical reading training course) can improve programming outcomes in introductory computing, encouraging additional work on understanding the cognitive basis of programming. Given the potential impact of cognitive interventions on pedagogy, we desire a better understanding of the cognitive processes underlying logical reasoning (i.e., *how students understand computational logic*). Unfortunately, a fundamental understanding of such cognitive processes is currently missing. As such, we restrict the scope of this dissertation proposal to obtaining a better understanding of the cognition behind logical reasoning, and offer suggestions for follow-on studies and interventions in the classroom.

To obtain an understanding of logical cognition, we desire a solution that satisfies the follow criteria:

- **Non-intrusive Methodology.** While approaches like Functional Near-Infrared Spectroscopy (fNIRS) and Functional Magnetic Resonance Imaging (fMRI) have been successfully used to investigate user cognition in Computer Science [19, 20, 31, 38, 80, 81], such approaches often carry ecological validity concerns (e.g., a programmer inside an fMRI machine or connected to fNIRS equipment is performing computer science tasks in a non-traditional environment, study stimuli may be limited to a 30 second duration, etc.) [31, Sec. VI]. We propose a solution that allows for user cognition to be investigated non-intrusively (i.e., in a more natural setting) with minimal interference to user attention.

- **Objective Measures.** Research from both Psychology and Computer Science has cast doubt on the trustworthiness of subjective measures [14, 22] (e.g., self-reporting). As such, we favor objective measures to understand cognition for computational logic in a generalizable way.

- **Context-specific Models.** We desire an understanding of a variety of comprehension tasks for computational logic. Recent research has shown that user cognition varies for different computer science activities (e.g., different regions of the brain are correlated with code comprehension [20, 30] and code writing [38]). Instead of producing one overly-generalized model for logical cognition, we propose to investigate context- or task-specific models associated with computational logic.

- **Incoming Preparation.** Software engineers entering a workforce often have different levels of incoming preparation or expertise [41], and this disparity in preparation is also reflected in students at the undergraduate level [71]. Since we aim to inform pedagogical interventions and follow-on investigations, we desire a solution that accounts for incoming preparation in the produced mathematical model (i.e., applies to students regardless of their levels of incoming preparation).

While previous research has investigated user cognition for computer science activities, such efforts have failed to satisfy one or more of the aforementioned desired properties. For instance, several neuroimaging studies investigating the cognitive processes behind programming have administered study stimuli inside an fMRI machine or with an array of external fNIRS connections attached to the participants [19, 20, 31, 38, 80, 81], calling into question the ecological validity of such experiments. By contrast, we desire an approach that allows the participants to complete the study tasks in a more traditional environment. Other studies investigating the psychology of programming (e.g., user behavior while programming [7, 59]) in more ecologically-valid settings often rely on self-reporting data or subjective measures that may not be adequately trustworthy [5]. Critically, many studies on user cognition for computer science tasks also do not shed light on how factors such as incoming preparation or expertise, among others, affect decision making in such programming tasks [13, 21, 89]. Since we aim to inform pedagogy based on student cognition, we desire an approach that explicitly accounts for incoming preparation in our models, and as such, can be applied to a wider variety of student groups.

We combine several insights to present a systematic study of student cognition for computational logic. **First**, we can implement objective, non-intrusive (i.e., natural) measures in a computer science context to establish correlations between variables. In particular, for this dissertation proposal, we hypothesize that with the emergence of technologies like high-precision eye-tracking, coupled with novel state-of-the-art fault localization approaches for hardware designs, it is possible to study user cognition for computation logic in a more ecologically-valid setting. **Second**, we can use medical devices in a computer science context to investigate causal relationships. In this proposal, we adapt the scientific approach and methodology behind transcranial magnetic stimulation (TMS) from psychology and medicinal research, and apply it in a software engineering context for the first time to establish causality in brain activity. **Finally**, we can use more advanced statistical rigor in computer science to account for student background and context. For this proposal, we hypothesize that accounting for incoming preparation or expertise as an explicit independent variable in our produced models can allow suggested pedagogical interventions to be applied to more students with different levels of preparation.

Combining these insights allows us to present the following three research components investigating cognition for computational logic:

(1) ***Does pointing programmers to the potential source of defects in a hardware design help their debugging efficacy?*** We propose to develop the first automated program repair (APR) tool for hardware designs (i.e., digital logic), and to investigate its use as a debugging aid for novice and expert programmers. Our human participants will be shown defective hardware designs with partial and full debugging hints (see Section 3.1), and we will compare their debugging efficacy against a baseline of no debugging information. We will test the utility of the tool as a debugging assistant using statistical tests on behavioral data collected from our participants. We hypothesize that our tool support can help improve the debugging efficacy of designers.

(2) ***How exactly do programmers locate incorrect parts of an algorithmic proof?*** We propose to use eye-tracking to gather insights into student problem-solving strategies for formalism comprehension tasks (i.e., mathematical logic). We further propose to investigate any differences in strategies (e.g., visual attention on different parts of the proof) used by students with different incoming preparation for formal methods. Our participants will be shown a series of algorithmic proofs, and will be asked to identify any mistakes in the proofs. We will use statistical tests on the eye-tracking, behavioral, and outcome data to compare the strategies employed by students with different levels of incoming preparation. We hypothesize that understanding the strategies used by different students and their outcome success rates can help shed light on how educators can better teach formalisms to undergraduate students.

(3) ***If we temporarily excite or inhibit the brain regions associated with spatial reasoning, does it impact programmers' ability to reason about code?*** We propose to investigate the relationship between spatial reasoning and program comprehension tasks (i.e., programming logic) using transcranial magnetic stimulation (TMS). Previous studies have found neurological correlations between spatial visualization and various program comprehension tasks [17,31], but a causal relationship between the two cognitive processes is yet to be established. We will use TMS to stimulate two brain regions associated with spatial reasoning tasks and one control region before presenting participants with program comprehension tasks. We will use statistical tests on the collected behavioral data to determine the presence of a casual link between brain activity for spatial reasoning and that for program comprehension. We hypothesize that such a causal link at the neurological level could spur changes in introductory programming education (e.g., including spatial visualization training) and improve student outcomes.

The overarching thesis statement of this proposal is:

> *It is possible to use objective measures to obtain mathematical models of the cognitive processes underlying logical reasoning, and these models can accurately explain student behavior.*

We hypothesize that constructing such models could help shed light on how educators can better teach logical reasoning to students, particularly those with less prior incoming preparation.

This proposal includes, to the best of our knowledge, the first automated program repair tool aimed at diagnosing and repairing hardware design defects at the circuit description level. Within a year of publication in a peer-reviewed venue, our preliminary results have been built upon by five novel studies on APR for hardware designs, and have resulted in discussions with a leading semiconductor manufacturer (Micron Technologies) focusing on adapting our circuit repair approach for internal use. We also propose the first TMS study of programming, demonstrating the applicability of the technique to computer science research. We have made publicly available the source code and datasets (including our statistical analysis scripts) for preliminary results associated with the first and second proposed research component. We will similarly make publicly available de-identified datasets for our third research component to support open and reproducible science.

The rest of this proposal is structured as follows: Section 2 outlines the background and related work, Section 3 describes the research and technical approach, Section 4 discusses our evaluation plans, Section 5 presents preliminary results for our research components, Section **??** outlines the schedule of our proposed dissertation, Section 7 highlights the broader impact of our work, and Section 8 concludes the proposal.

2 Background and Related Work

In this section, we outline the background and related work for our proposed research.

2.1 Automatic Error Diagnosis and Correction

In modern engineering, the hardware design process typically includes producing digital specifications (often using hardware description languages, or HDLs) for electronic devices, computer systems, or integrated circuits that allow for the creation of physical hardware products [45]. Hardware or HDL designs differ from software programs in two key ways. First, hardware designs are inherently parallel and often include non-sequential statements, since separate portions of hardware can operate simultaneously. While some conventional languages, such as Javascript, have support for parallelism, software written in languages such as C and Java is generally based around a serial execution model. Second, software programs usually use

test cases to evaluate functional correctness, where individual test cases may pass or fail depending on the quality of the software. HDL designs, on the other hand, use *testbenches* [45], which are programs with documented and repeatable sets of stimuli, to simulate behaviors of a device under test (DUT).

While a significant amount of work has been done in automatic error diagnosis of hardware designs, the correction of such errors automatically has not been well-explored to the best of our knowledge. Techniques in the works of Jiang *et al.* [32] and Ran *et al.* [65] employ software analysis approaches to identify statements in design code responsible for defects, but suffer from high false positive rates. Bloem and Wotawa [9] use formal analysis of circuit descriptions to identify defects, and Peischl and Wotawa [60] use a model-based diagnosis paradigm that supports source-level debugging of large industrial designs at the statement and expression level. This use of formal methods for error diagnoses is orthogonal to our work, but could be applied to reduce the search space for our proposed approach. Staber *et al.* [85] use state-transition analysis to diagnose and correct hardware designs automatically, but their techniques similarly do not scale to real-world circuits with large state spaces. Our proposed approach, by contrast, is more scalable to larger, real-world hardware descriptions.

In the realm of software, significant research effort has been devoted to repairing bugs automatically over the last decade [23,43,52]. Automated program repair usually takes as input source code with a deterministic bug and a test suite with at least one failing test that reveals the bug, and aims to automatically generate fixes to the buggy code. Test suite based repair, where test cases are used to guide the search for a patch, can be further divided into generate-and-validate and semantics-driven approaches. Generate-and-validate techniques produce candidate patches for the buggy code and evaluate them against the test suite to check if all tests pass [2, 39, 62, 63]. Semantics-driven approaches first extract constraints on a program based on test suite execution and then use these constraints to synthesize a patch [47–49, 55]. While software approaches to APR make use of test suites to evaluate candidate repairs, our proposed approach uses instrumented hardware testbenches to make visible the internal and external behavior of a simulated circuit for fitness evaluation. Additionally, APR for software usually uses fault localization techniques that do not support the parallelism exhibited by hardware.

2.2 Eye-tracking and Cognition

Modern eye-tracking cameras measure and track a participant's eyes and use event detection algorithms to report *gaze data* that is then analyzed with respect to pre-defined *areas of interest* (AOIs) in a stimulus. AOIs are typically manually defined by an experimenter based on the nature of the study [24, 77]. A *fixation* is an eye gaze that lasts for approximately 200–300ms on a specific AOI and results in the focus of visual attention on the AOI. The majority of information processing for humans occurs during fixations [25, 34] and a small number of fixations usually suffices for a human to process a complex visual input [24, 66]. As such, fixation data is widely used to measure cognitive load for different tasks, with longer fixations and higher number of fixations indicating higher cognitive load [76, 78]. The *attention switching* metric depends on fixation counts and measures the total number of switches between AOIs, and can approximate the dynamics of visual attention during a task [76].

Previous eye-tracking work investigating problem-solving strategies employed by users has shown that differences in search strategies can lead to significant differences in task outcomes [26, 54, 87]. For instance, Netzel *et al.* found that high-accuracy users were better able to use information in science-related diagrams [54]. Hegarty *et al.* [26] investigated arithmetic problem solving involving relational terms inconsistent with the required arithmetic operator (e.g., the use of *less than* for tasks involving addition), and found that low-accuracy users made more reversal errors for inconsistent problems, and that high-accuracy ones required more re-readings for previous text fixations. We propose to similarly investigate the cognitive

loads and the corresponding outcome successes in participants fixating over different aspects of a formalism presentation.

Figures are frequently used as educational instruments [11, 46, 92], yet their importance as a medium of instruction for a particular field is not always well-understood. For instance, Susac *et al.* [87] found that diagram were rarely helpful for physics. By contrast, Yoon *et al.* [93] studied the importance of figures for causal reasoning problems, and found that even for questions missing a figure, 48% of the students still frequently fixate on the area where the figure would have been, indicating a relative higher importance for figures. For both studies, however, the inclusion of diagrams did not affect the participants' time taken to respond or response accuracy. In a mistake-finding context for formalism comprehension, we similarly propose to investigate relationship between fixation on, or perceived importance of, figures on task outcomes.

2.3 Medical Imaging, Transcranial Magnetic Stimulation and Cognition

In recent years, the software engineering community has increasingly used medical neuroimaging to understand the cognitive processes behind programming [19, 20, 31, 38, 80, 81]. Such studies have identified cognitive processes correlated with various software engineering tasks. For example, many of the neuroimaging studies in software engineering have found connections between programming and reading [20] or spatial visualization [18, 31], two skills with well-understood cognitive structures. Note, however, that medical imaging studies often only provide confidence in *correlations* between variables, not causative links. Other studies comparing pedagogical effects of cognitive interventions suggested by correlations (e.g., spatial training [10, 17, 56, 83]) have shown the potential to improve programming outcomes in introductory computing, encouraging additional work on understanding the cognitive basis of programming.

Transcranial Magnetic Stimulation (TMS) is a safe and noninvasive technique that is well-established for a variety of clinical and scientific use cases [88]. When administered, TMS produces magnetic fields which stimulate (or disrupt) a region of the brain by inducing an electric current in the neurons of this region [88]. Clinically, TMS is used as a treatment for major depressive disorder [82], smoking cessation [15], and obsessive-compulsive disorder [91], among others. TMS is a well-studied research tool: in the past 10 years, the National Library of Medicine has recorded over 1000 academic papers published each year which investigate the use of TMS. Compared to other methods, TMS is a time-efficient way to investigate the *causative* link between neural activity and programming ability. Other medical approaches that affect the brain in specific areas tend to be quite invasive, requiring implanted electrodes, drug treatments, or neurological surgeries. By contrast, non-medical approaches such as transfer training or pedagogy are typically studied over a longer period of time (*cf.* [17]). Using TMS also removes variance and potential confounds which may be present in a study extending over a long period of time (e.g., changing physical or mental states of participants). By disrupting brain activity using this neurostimulation technique and then measuring behavioral outcomes (e.g., timing, accuracy, etc.) on programming tasks, we can observe a causal relationship between spatial reasoning and programming ability, should one exist.

3 Proposed Research

The goal of this proposal is to provide a set of systematic studies that objectively measure user cognition for reasoning about computational logic in software engineering. We further propose to view logical reasoning through three interrelated lenses: digital logic, mathematical logic, and programming logic. We propose the following three research components to achieve this systematic understanding:

1. Developing an automated program repair tool for hardware designs (i.e., digital logic) and a mathematical model of the effects of its use as a debugging assistant for programmers
2. Developing a mathematical model of the cognitive processes for formalism comprehension (i.e., mathematical logic) using eye-tracking
3. Developing a mathematical model for the connection between program comprehension (i.e., programming logic) and spatial reasoning using transcranial magnetic stimulation (TMS)

The rest of this section describes each research component at a high level.

3.1 Research: Automated Program Repair for Hardware Designs

In the first research component, we aim to investigate the use of an automated program repair (APR) tool for hardware designs (i.e., digital logic) as a debugging aid for novice and expert programmers.

Given the lack of existing automated tools to diagnose and repair defects in hardware designs (see Section 2.1), we propose to develop CirFix, a framework for automatically repairing defects in hardware designs implemented in languages like Verilog, one of the most popular HDLs [35]. CirFix uses an iterative stochastic search based on readily-available artifacts in the hardware design process (e.g., testbenches, simulation environments) to diagnose and repair defects in a circuit description. We propose to guide the search for a repair by instrumenting hardware testbenches to record the values of output wires at specified time intervals during circuit simulations. CirFix then performs a bit-level comparison of output wires against information for expected behavior to assess functional correctness of candidate repairs. CirFix employs a fixed point analysis of assignments made to internal registers and output wires to implicate statements and reduce the search space, enabling our approach to scale to larger circuit designs.

We further propose to conduct a human study using this novel fault localization algorithm as a debugging aid for real-world and student circuit designs. To investigate the incremental benefit of our fault localization, we will consider three scenarios: the full output of the algorithm, only initially implicated statements of the algorithm (without any transitive information, see [3, Algorithm 1]), and no fault localization annotations. We will ask participants to: (1) identify faulty lines in each design shown, (2) indicate which lines they would alter to fix each defect, and (3) propose how they would alter the lines to fix each defect if they could patch it.

3.2 Research: Cognition for Formalisms

In the second research component, we propose to use eye-tracking to gather insights into (i) the problem-solving strategies for such formalism comprehension (i.e., mathematical logic) tasks employed by students with different levels of familiarity with, or *incoming preparation* for, formal methods, and (ii) how educators can better prepare struggling students for the rigorous logical reasoning required by high-assurance software engineering.

We hypothesize that understanding how people less familiar with formalisms think about formal methods and proofs of algorithmic properties is critical to how educators should teach formalisms. For instance, since the most vulnerable population groups with non-traditional backgrounds are also most likely to drop the computer science major [37], educators may favor making sure the needs of such groups are not overlooked. It also indirectly impacts how high-assurance software engineering firms might train new workers. One way to acquire this understanding is through the investigation of the *cognition* (e.g., problem-solving

strategies, cognitive load, visual attention, etc.) for computer science students while performing formalism comprehension tasks.

We propose to perform a controlled experiment investigating how students read and assess formal proofs about algorithms for correctness. We will recruit students with varying levels of expertise with formalisms to perform these comprehension tasks. Participants will be presented with pseudocode algorithms from a widely-used undergraduate textbook [68], a theorem about each algorithm with an accompanying formal proof, and a graphical illustration of the algorithm or proof. Participants will then be asked to evaluate the presented proofs for correctness.

3.3 Research: Programming and Spatial Reasoning

In the third research component, we propose to codify the relationship between spatial reasoning and program comprehension tasks (i.e., programming logic) using transcranial magnetic stimulation (TMS). While previous studies have found neurological correlations between spatial visualization and various program comprehension tasks [17, 31], a causal relationship between the two cognitive processes has yet to be confirmed.

We hypothesize that confirming a causal link between spatial ability and program comprehension at the neurological level could spur changes in introductory programming education (e.g., including spatial visualization training, *cf.* [10, 56, 83]) and improve student success. Furthermore, since this is, to the best of our knowledge, the first TMS study of programming, we aim to demonstrate the applicability of TMS to computer science research.

We propose to perform a controlled human study investigating potential causal links by using TMS to disrupt visualization-associated regions of the brain and assessing participant performance on programming tasks. The experiment will conducted in two parts: an fMRI scan followed by two to four subsequent TMS sessions (each hosted on a different day). During the fMRI scan, participants will be shown three general types of stimuli: data structure manipulation, code comprehension, and mental rotation. For TMS sessions, participants will be subjected to two experimental conditions (stimulation of the pre-motor and SMA region, and stimulation of the primary motor cortex) and one control condition (stimulation of the cranial vertex region). Both experimental regions are associated with spatial reasoning (*cf.* [12]), while the control region is associated with leg movement [33]. After TMS, participants will be tested on our stimuli. Individual stimuli will not be shown more than once. This experimental design will allow for a controlled investigation of the potential causal link between spatial reasoning and program comprehension.

4 Proposed Experiments and Metrics

In this section, we discuss our experimental design and metrics for evaluating each proposed research component. An appropriate handling of false discovery rates, effect sizes and statistical significance will be made throughout our mathematical analyses across all research questions.

4.1 Experimental Design: Automated Program Repair for Hardware Designs

Our first research component proposes to develop CirFix, an APR tool for hardware designs, and investigate its use as a debugging aid for designers. We aim to answer the following research questions: (1) What fraction of defect scenarios in our benchmark suite can CirFix repair? (2) Does CirFix's fault localization algorithm improve designers' objective performances? (3) In what contexts do designers find the CirFix fault localization algorithm helpful?

RQ1. For our evaluation of CirFix, we desire a benchmark suite consisting of faulty hardware designs that are indicative of defects in industry, comprise a wide range in terms of project size, and correspond to a variety of components found in real-world designs. To the best of our knowledge, there are no publicly-available benchmarks that satisfy our requirements. Additionally, there is limited open source community support for industrial hardware designs, since such designs are often considered Intellectual Property (IP) of the stakeholder companies. As such, we propose to adapt the defect-seeding approach common in software [16, 55, 75] and present a benchmark suite of *defects scenarios* [39, 40] to be used for the evaluation of automated repair techniques for hardware. We will run multiple trials of CirFix with resource and time bounds. We will consider our approach to be successful if it can achieve a repair rate comparable to those of common software APR approaches (e.g., 52.4% from GenProg [39]).

RQs 2 and 3. To investigate the usability of our novel fault localization algorithm as a debugging aid independent of the automated repair context, we propose to conduct a controlled human study in which we ask students to assess its quality and usefulness. We will assess programmer performance by evaluating (1) F-scores (F_1) of correctly-identified faults for each debugging task by each participant and (2) total time taken to complete a debugging task within no specific time limit. A participant is said to correctly identify faults for a given defect scenario if they identified program line(s) that contain a bug or missing line(s). F-scores will be evaluated by calculating the harmonic mean of recall and precision. To evaluate the statistical significance of participants using the fault localization as a debugging aid as opposed to none, we will use the unpaired Student t-test. To avoid false positives as a result of repeated analyses, we will apply a *false discovery rate* (FDR) correction ($q < 0.05$). We will consider our approach successful if we obtain a statistically significant difference ($p < 0.05$) in the F-scores and debugging time for programmers using our tool when compared against a baseline.

4.2 Experimental Design: Cognition for Formalisms

Our second research component proposes to use eye-tracking to investigate cognition for formalism comprehension tasks. We aim to answer the following research questions: (1) What is the effect of incoming preparation on student outcomes for our formalism comprehension tasks? (2) How do student self-reports of formalism comprehension tasks align with more objective empirical results? (3) What factors most distinguish higher-performing students from lower-performing ones?

RQ1. We will consider both prior coursework and current performance in our definition of incoming preparation. We will first ask participants to outline the number of computer science theory courses covering formalisms (e.g., derivations and proofs) they have either completed with passing grades or are currently taking. We will then screening participants by testing them on a mistake in a proof taken from an undergraduate textbook [68]. We propose to partition our sample based on whether a participant has taken more than the median number of courses in our sample *and* passed the screening test. Both facets of incoming preparation that we consider have been used previously in the context of pedagogy to approximate incoming preparation [71, 90]. We will use the Mann-Whitney U-test to investigate the response accuracy and time between more-prepared and less-prepared participants, and interpret a statistically significant difference ($p < 0.05$, FDR-corrected) as effectively establishing the role of incoming preparation on the outcomes of formalism comprehension.

RQ2. We will instruct participants to provide answers to a post-questionnaire reflecting on their experiences with the study and outlining what they thought to be the most important parts of a formalism presentation. In addition to having participants self-report (on a 1–5 Likert scale) the difficulty of tasks they were asked to perform, the helpfulness of different aspects of the formalism presentation, and so on, we will ask three free-response questions: (1) Having completed the study session, would you do anything different the next time around? (2) What is the most important thing that makes a proof easier to understand? (3) What is the most important thing that makes it easier to spot a mistake in the proof? To understand the relationship between participant Likert scale responses and response accuracy, we will use the Kendall's τ test to conduct a quantitative analysis of the data. We will consider values of $|\tau| \geq 0.49$ as strong evidence of a correlation, $0.26 \leq |\tau| < 0.49$ as moderate evidence of a correlation, and $|\tau| < 0.26$ as no or weak evidence of a correlation [74].

RQ3. We are interested in investigating the factors that cause students to perform better at formalism comprehension tasks. To do so, we will perform a sub-population analysis of students with higher and lower outcomes. We will require a participant to have achieved above the median response accuracy to be classified as higher performing, with the remaining participants considered lower performing. We will use a χ^2 test to compare the response accuracies of higher and lower performing participants for different categories of proofs and algorithms. Following the Goldberg and Helfman guidelines [24] for defining AOIs in terms of size and granularity, we will manually divide each presented stimulus into six AOIs: *Algorithm*, *Theorem*, *Proof*, *Figure*, *Correct Answer*, and *Distractors*. We will use the Mann-Whitney U-test to compare the problem solving strategies employed by higher and lower performing participants (measured using gaze fixation metrics for different AOIs), with a value of $p < 0.05$ indicating a statistically significant difference in strategies employed by the more successful participants.

4.3 Experimental Design: Programming and Spatial Reasoning

Our final research component proposes to use TMS to investigate the relationship between spatial reasoning and program comprehension. We aim to answer the following research questions: (1) Does disrupting brain regions associated with spatial reasoning (i.e., the pre-motor / SMA region and the primary motor cortex) affect a programmer's ability to correctly perform program comprehension tasks (e.g., manipulating data structures, tracing through code, etc.)? (2) Does disrupting brain regions associated with spatial reasoning affect the time taken for a programmer to perform program comprehension tasks? (3) How does demographic information (e.g., incoming preparation or expertise) mediate the effect of TMS on task outcomes?

RQs 1 and 2. Since we will not be analyzing brain imaging data to address our research questions, we will be primarily using standard frequency-based statistical approaches (e.g., paired t-test, ANOVA) to compare the accuracy and reaction times (i.e., behavioral outcomes) for both the spatial-reasoning and programming tasks in all three TMS stimulation conditions. Our null hypothesis will be rejected if there are significantly different times or accuracies in either of the two spatial-region stimulation sessions for either the programming or spatial tasks compared to the active control session. We will additionally control for multiple comparisons using the Benjamini-Hochberg (BH) adjustment.

RQ3. To investigate the effects of TMS on task outcomes for study subjects with different levels of expertise, we will perform a sub-population analysis of participants with higher and lower outcomes on the baseline (i.e., control region) TMS session. We will require a participant to have achieved above the median

response accuracy to be classified as higher performing. We will then use the χ^2 test to investigate the effects of TMS on the response accuracy and time for the higher and lower performing participants, with a value of $p < 0.05$ indicating a statistically significant difference in the effects of TMS on participants with different levels of expertise.

5 Preliminary Results

This section presents preliminary results (where applicable) for our proposed research components.

5.1 Preliminary Results: Automated Program Repair for Hardware Designs

We have completed the experiments proposed for this research component in Section 4.1. We conducted a human study involving 41 participants to answer RQs 2 and 3 (IRB: HUM00199335). Results for each RQ are summarized below. Preliminary results were published in the International Symposium on Architectural Support for Programming Languages and Operating Systems in 2022 [3], with follow-on journal work accepted in the Transactions of Software Engineering in 2023 [72].

RQ1: Repair Rate and Quality. Table 1 presents the repair results for each defect scenario. CirFix produced *plausible* (i.e., testbench-adequate) repairs for 21 of the 32 (65.6%) defects in our benchmark suite. Of the 11 defects that were not repaired, 4 exhausted resource limits while 7 required edits not supported by CirFix operators and repair templates. While a direct comparison between CirFix and APR for software is not possible, we observe that the repair rate of CirFix comparable to the reported repair rates of well-known software repair approaches, e.g., GenProg (52.4%) [39], Angelix (34.1%) [49], and TBar (53.1%) [42].

We follow the approach taken by Long and Rinard [44] for patch assessment since it follows best practices in the APR literature [61, 63]. We manually analyze the 21 repairs produced by CirFix. We found 16 of the generated repairs to exhibit correct behavior, with the final 5 to be correct only with respect to the testbench (i.e., overfitting). While room for improvement remains, software industrial deployments with similar rates have proved useful: for example, Bloomberg reported that a 48% correct patch rate was associated with "very positive" feedback and a general "helpful" opinion [36, p. 5].

RQ2: Fault Localization and Human Performance. We did not observe a statistically-significant difference in time taken to localize faults with full or no annotations from our fault localization ($p = 0.41$, Student t-test). On

Table 1: Preliminary results for CirFix. A missing time repair was found in 5 independent trials. CirFix produced plausible repairs to 21 of the 32 defect scenarios, of which 16 were correct upon manual inspection (denoted by ✓).

Project	Defect Description	Repair Time (s)
decoder_3_to_8: 3-to-8 decoder	Two separate numeric errors	✓ 13984.3
	Incorrect assignment	—
counter: 4-bit counter with overflow	Incorrect sensitivity list	✓ 19.8
	Incorrect reset	✓ 32239.2
	Incorrect incremental of counter	✓ 27781.3
flip_flop: T-flip-flop	Incorrect conditional	✓ 7.8
	Branches of if-statement swapped	✓ 923.5
fsm_full: Finite state machine	Incorrect case statement	—
	Incorrectly blocking assignments	4282.2
	Assignment to next state and default in case statement omitted	1536.4
	Assignment to next state omitted, incorrect sensitivity list	✓ 37.0
lshift_reg: 8-bit left shift register	Incorrect blocking assignment	✓ 14.6
	Incorrect conditional	✓ 33.74
	Incorrect sensitivity list	✓ 7.8
mux_4_1: 4-to-1 multiplexer	1 bit instead of 4 bit output	—
	Hex instead of binary constants	10315.4
	Three separate numeric errors	15387.9
i2c: Bidirectional serial bus	Incorrect sensitivity list	✓ 183
	Incorrect address assignment	57.9
	No command acknowledgement	✓ 1560.5
sha3: Cryptographic hash function	Off-by-one error in loop	✓ 50.4
	Incorrect bitwise negation	—
	Incorrect assignment to wires	—
	Skipped buffer overflow check	✓ 50.0
tate_pairing: Tate bilinear pairing for elliptic curves	Incorrect logic for bitshifting	—
	Incorrect operator for bitshifting	—
	Incorrect instantiation of modules	—
reed_solomon_decoder: Reed-Solomon error correction	Insufficient register size for decimal values	—
	Incorrect sensitivity list for reset	✓ 28547.8
sdram_controller: Synchronous DRAM controller	Numeric error in definitions	—
	Incorrect case statement	—
	Incorrect assignments to registers during synchronous reset	✓ 16607.6

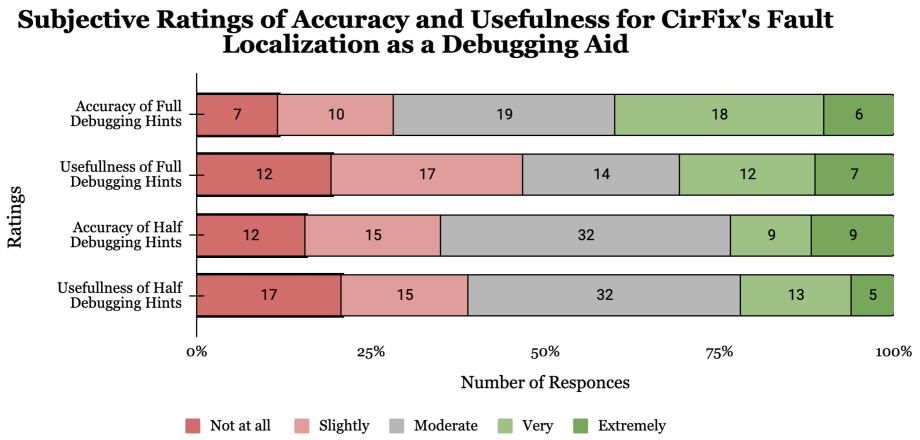


Figure 1: A visual representation of the distribution of ratings subjects gave to CirFix's fault localization when viewed as a debugging aid. Subjects rated the tool as a debugging aid based on accuracy and usefulness on a scale of 1–5, where 1 represents not at all accurate or useful and 5 represents extremely accurate or useful.

average, participants spent 299.6 seconds with full annotations as opposed to 239.0 seconds with no annotations. We did find that the objective F-score for participants given full localization was higher ($F_1 = 0.67$) than the objective F-score for participants who had half fault localization ($F_1 = 0.33$), which in turn was higher than those without fault localization ($F_1 = 0.29$).[1] However, this trend did not rise to the level of statistical significance ($p = 0.12$). We predict that the results indicate our proposed notion of fault localization can be a useful tool for manual debugging.

In addition, we found statistically-significant differences in the F-scores between experts ($F_1 = 0.37$) and novices ($F_1 = 0.17$) when they had CirFix's fault localization with a large effect size ($p = 0.04$, $d = 0.54$), but this statistic did not survive correcting for multiple comparisons.

RQ3: Subjective Judgment of Fault Localization. For each presented stimulus with a debugging aid, participants were asked to rate, on a Likert scale, the usefulness and accuracy of the tool in helping them localize the circuit defects as seen on Figure 1. Differences in the number of responses per rating arise because not all participants answered all questions.

Participants rated full fault localization support on student-developed designs to be significantly more useful and accurate than full support for open source projects ($p = 0.01$, $d = 0.7$; $p = 0.002$, $d = 1.05$, a large effect size). These results suggest our algorithm would be more beneficial for debugging in pedagogical environments.

Most interestingly, we find that participants rated CirFix's fault localization support to be significantly more useful and accurate for debugging multi-line defects than single-line defects with a large effect size ($p = 0.002$, $d = 1.04$; $p = 0.003$, $d = 0.86$). Given that support for multi-line software repairs is limited [49, 70], with most tools only supporting single-line repairs, our results, by contrast, are promising for reducing maintenance costs associated with more complex defects in the hardware domain.

[1] A participant with an F-score of 1 correctly identified faulty program line(s) or missing line(s) in the defect scenario, while a F-score of 0 meant no faulty program line(s) or missing line(s) were correctly identified.

5.2 Preliminary Results: Cognition for Formalisms

We conducted a human study involving 34 participants to complete the experiments proposed for this research component in Section 4.2 (IRB: HUM00204278). Results for each RQ are summarized below. Preliminary results were published in the International Conference on Software Engineering in 2023 [4].

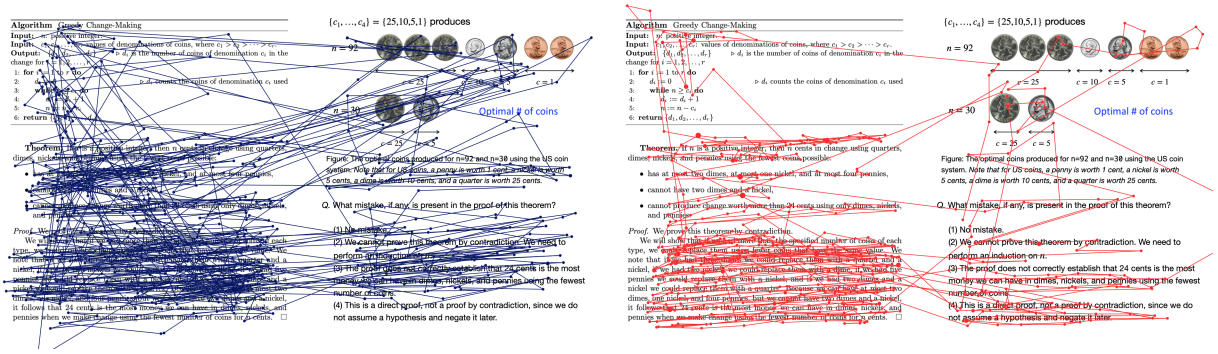
RQ1: Role of Incoming Preparation. We found no evidence of a statistically-significant difference in the outcomes between more-prepared and less-prepared students, both in terms of response accuracy (two-tailed Mann-Whitney U-test with the Benjamini-Hochberg [8] procedure to correct for false discoveries, $p = 0.96$) and response time ($p = 0.93$). Notably, while absence of evidence is not evidence of absence, the effect sizes for both results were extremely small (Cohen's $d = 0.007$ for response accuracy and $d = 0.08$ for response time), giving statistical confidence in the null result (i.e., even if an effect were present, it would be of very low magnitude and thus unlikely to influence outcomes). We further found no correlation between the number of theory courses taken and response accuracy (Pearson's $r = 0.036, p = 0.84$), nor a correlation between participants' self-perceived experience with formalisms (on a 1–5 Likert scale) and response accuracy (Kendall's $\tau = 0.21, p = 0.18$).

Our results indicate that, contrary to conventional wisdom [67, 71, 86, 90] and instructor expectations, students with greater incoming preparation perform no better at these formalism tasks, on average, than students with lower incoming preparation. Indeed, the two participants with the highest number of courses taken had the lowest and second-lowest response accuracies.

Even though participants have similar final outcomes, they employ different strategies. An analysis of visual behaviors between students with different incoming preparations reveals that more-prepared students *fixate longer* on (i.e., spend more time looking at) AOIs corresponding to the proof (two-tailed Mann-Whitney U-test with the Benjamini-Hochberg procedure, $p = 0.005$), correct answer ($p = 0.038$), and distractor answer choices ($p = 0.03$). Our results suggest that while incoming preparation teaches students to read a proof and the answer choices thoroughly before selecting an answer, this increased attention to the AOIs does not actually help students achieve better outcomes.

RQ2: Self-Reporting and Formalism Comprehension Tasks. To better understand the relationship between participant Likert scale responses and *response accuracy*, we use the Kendall's τ test to conduct a quantitative analysis of the data. When analyzing participants' *self-reported experience* with formalisms, we found no evidence of a correlation between experience and response accuracy ($\tau = 0.21, p = 0.18$). We also observed no correlation between *self-reported task difficulty* and study outcomes ($\tau = 0.14, p = 0.35$), nor a correlation between the *self-reported helpfulness of figures* for formalism comprehension tasks and study outcomes ($\tau = -0.22, p = 0.13$). Our results do not provide evidence that students are accurate at self-reporting their experience with formalism comprehension tasks (*cf.* unreliable self-reporting, see Section 1).

To further investigate whether student self-perception is an accurate predictor of factors associated with high outcomes, we also performed a qualitative analysis on the participants' self-reported free-response data. 26 out of 34 participants indicated they would employ a different problem-solving strategy if asked to do the study again. Tied for the most common change in strategy were paying more attention to the algorithmic pseudocode and reviewing the materials from core theory courses prior to the study. Our experimental results, on the other hand, do not indicate a relationship between *fixation time* on algorithmic pseudocode (i.e., time spent reading pseudocode) and higher response accuracy ($p = 0.91$). The desire to review materials from core theory courses is aligned with our experimental results: students with greater incoming preparation do not perform better, suggesting a lack of retention of course materials over time.



(a) Higher-outcome participant gaze plot

(b) Lower-outcome participant gaze plot

Figure 2: Visual gaze plots for a stimulus shown to two participants. The higher-outcome participant (left) displays significantly more attention switching behaviors, as indicated by the number of lines crossing between different AOI quadrants.

When asked to describe the features that make a proof easier to comprehend, about a third of the participants mentioned concise, easy-to-read English prose in the proof. The second most popular answer (6/34 participants) corresponded to the use of figures and visuals while reading the proof. Interestingly, our empirical results do not show evidence of a significant correlation between *self-perceived proof readability* and outcomes for formalism comprehension tasks (Kendall's $\tau = -0.14, p = 0.32$), or a statistically significant relationship between increased *fixation* on (or attention to) figures and response accuracy ($p = 0.81$).

RQ3: Factors Associated with Higher Outcomes. Given the apparent lack of effect of incoming preparation on the outcomes for our study, we are interested in investigating the factors that result in students performing better at formalism comprehension tasks. To do so, we perform a sub-population analysis of students with higher and lower outcomes. We require a participant to have achieved above the median response accuracy to be classified as higher performing.

We examined the outcomes for higher and lower performing students for different proof categories (inductive, contradictory, and direct) and algorithm categories (recursive, iterative, and non-repeating). We found that, independent of algorithm category, higher performing students are more likely to spot mistakes in proofs by induction than the lower performing ones (χ^2 test with the Benjamini-Hochberg procedure, $p = 0.01$). We also find that independent of problem or proof type, higher performing students are more likely to get proofs for recursive algorithms correct compared to lower performing students ($p = 0.006$).

Recall that *attention switching* measures the total number of switches between AOIs, and can approximate the dynamics of visual attention during a task. We found that higher performing students demonstrate more *attention switching* behaviors, or frequently go back and forth between AOIs on the presented materials (two-tailed Mann-Whitney U-test with the Benjamini-Hochberg procedure, $p = 0.002$). In particular, we observed a statistically-significant difference in attention switching for proofs by contradiction ($p = 0.009$) and iterative algorithms ($p = 0.007$). Figure 2 shows two illustrative visual gaze plots for a higher outcome and another lower outcome participant for a stimulus involving proof by contradiction. The higher-outcome participant displays significantly more attention switching behavior (as indicated by the increased number of lines between the AOI quadrants, see Figure 2a) compared to the lower-outcome participant (Figure 2b). In the context of pedagogy, these results argue for the development of teaching materials, such as online tools, lecture slides, and exams, that facilitate perusal with ease (e.g., without requiring multiple page flips).

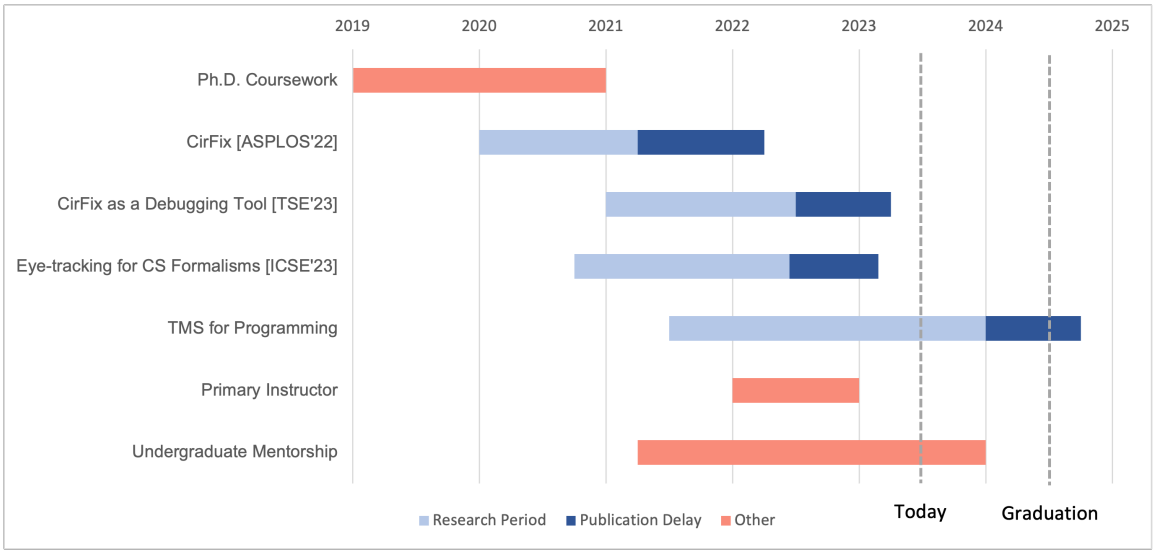


Figure 3: Proposed dissertation work schedule.

5.3 Preliminary Results: Programming and Spatial Reasoning

We have obtained an internal grant of $17,070 from the University of Michigan Functional MRI Laboratory to fully cover the costs associated with this study, completed the stimulus design for the study, and obtained IRB approval (IRBMED: HUM00216195). Additionally, we have completed the safety training required to conduct TMS sessions, and have obtained research access to the necessary fMRI and TMS equipment. A preliminary power analysis suggests that we would need $n \geq 15$ participants for statistically significant results [12]. Participant recruitment is currently on-going for this study, and we plan on completing recruitment and doing data analysis in the Fall'23 and Winter'24 semesters respectively.

6 Schedule

Figure 3 outlines the timeline for the proposed research. We anticipate completing the remaining research efforts in 1 year, with an expected graduation date of May 2024.

We have previously targeted venues such as the *International Symposium on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), the *International Conference on Software Engineering* (ICSE), and *Transactions on Software Engineering* (TSE). We propose targeting similar venues for the remaining work, as well as other venues dedicate to computer science education, such as the *Technical Symposium on Computer Science Education* (SIGCSE).

7 Broader Impact

In this section, we discuss the potential broader impact of our proposed research, with an emphasis on mentorship and pedagogy.

Mentorship. Since mentorship remains a primary goal of this dissertation, we will actively involve undergraduate students in the proposed research agenda. We are especially interested in mentoring students from

groups typically underrepresented in computer science. For instance, Zachary Karas is a returning adult student who contributed to, and was an author for, the peer-reviewed publication for preliminary results for the study involving eye-tracking (ICSE'23). Karas is now a Ph.D. student at Vanderbilt University working on more eye-tracking studies. Similarly, Priscila Santiesteban is a first-generation, Hispanic pre-doctoral student who has contributed to, and was a first author on, a peer-reviewed publication on hardware debugging assistance. We have mentored an additional two undergraduate students (including a woman student) on research included in this proposal. PI Ahmad has further written text for an NSF REU ($8,000 total awarded) to fully support an additional undergraduate student to assist with the proposed research for this dissertation.

Pedagogy. Informing pedagogy based on user cognition is a first-order desire for our proposed research. While studying cognition has already resulted in pedagogical interventions for various disciplines [28, 51, 53, 79], such studies have yet to influence how educators teach computer science undergraduate courses at large. Our proposed research will include recommendations for educators and suggest future intervention studies investigating those recommendations. PI Ahmad has already been approached by an educator at the University of Washington for a collaboration on using preliminary results from the second research component in this proposal (eye-tracking and computer science formalisms) to better teach undergraduate theory. Similarly, PI Ahmad has had conversations with a professor at the University of Michigan on deploying debugging assistance tools at the classroom level. Our proposed research also aims at broadening participation in computer science by directly accounting for incoming preparation as an independent variable, and thus has the potential to be applied to a wider variety of student groups.

8 Conclusion

Computational logic reasoning is ubiquitous in software engineering activities and forms a core component of undergraduate computer science curricula. However, we lack a fundamental understanding of the cognitive processes underlying logical reasoning. In this dissertation proposal, we propose a systematic approach to study cognition for three facets of computational logic: *digital logic* (e.g., hardware designs), *mathematical logic* (e.g., proofs about algorithms), and *programming logic* (e.g., coding, manipulating data structures). We hypothesize that it is possible to use objective measures to obtain mathematical models of the cognitive processes underlying logical reasoning, and these models can accurately explain student behavior. Obtaining an understanding of logical cognition could impact how educators teach logical reasoning to students with varying levels of incoming preparation or expertise.

In this proposal, we include three research components. First, we propose to develop the first-of-its-kind automated program repair (APR) tool for hardware designs (i.e., digital logic) and to investigate its use as a debugging aid for novice and expert designers. Second, we propose to use eye-tracking to gather insights into how students with different levels of incoming preparation understand algorithmic proofs (i.e., mathematical logic), and what educators can do to better teach formalisms to students. Third, we propose to codify the relationship between spatial reasoning and program comprehension (i.e., programming logic) using transcranial magnetic stimulation (TMS). This research will improve our understanding of student cognition behind logical reasoning and offer suggestions for follow-on studies and interventions in the classroom.

References

[1] ABET. *2022-2023 Criteria for According Computing Programs*. 2021. Accredication Board for Engineering and Technology, `https://www.abet.org/wp-content/uploads/2022/03/2022-23-CAC-Criteria.pdf`.

[2] T. Ackling, B. Alexander, and I. Grunert. Evolving patches for software repair. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1427–1434, 2011.

[3] H. Ahmad, Y. Huang, and W. Weimer. Cirfix: Automatically repairing defects in hardware design code. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2022, page 990–1003, New York, NY, USA, 2022. Association for Computing Machinery.

[4] H. Ahmad, Z. Karas, K. Diaz, A. Kamil, J.-B. Jeannin, and W. Weimer. How do we read formal claims? eye-tracking and the cognition of proofs about algorithms. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 208–220. IEEE, 2023.

[5] G. Alarcon and T. Ryan. Trustworthiness perceptions of computer code: A heuristic-systematic processing model. In *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[6] I. Barland, M. Felleisen, K. Fisler, P. Kolaitis, and M. Y. Vardi. Integrating logic into the computer science curriculum. In *Annual Joint Conference on Integrating Technology into Computer Science Education*, 2000.

[7] R. Bednarik and M. Tukiainen. An eye-tracking methodology for characterizing program comprehension processes. In *Proceedings of the 2006 symposium on Eye tracking research & applications*, pages 125–132, 2006.

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

[9] R. Bloem and F. Wotawa. Verification and fault localization for vhdl programs. *Journal of the Telematics Engineering Society (TIV)*, 2:30–33, 2002.

[10] R. Bockmon, S. Cooper, W. Koperski, J. Gratch, S. Sorby, and M. Dorodchi. A CS1 spatial skills intervention and the impact on introductory programming abilities. In *Technical Symposium on Computer Science Education*, pages 766–772, 2020.

[11] C. Buckley and C. Nerantzi. Effective use of visual representation in research and teaching within higher education. *International Journal of Management and Applied Research*, 7(3):196–214, 2020.

[12] G. Cona, G. Marino, and C. Semenza. Tms of supplementary motor area (sma) facilitates mental rotation performance: evidence for sequence processing in sma. *Neuroimage*, 146:770–777, 2017.

[13] T. Crews and J. Butterfield. Gender differences in beginning programming: an empirical study on improving performance parity. *Campus-Wide Information Systems*, 20(5):186–192, 2003.

[14] N. Dell, V. Vaidyanathan, I. Medhi, E. Cutrell, and W. Thies. " yours is better!" participant response bias in hci. In *Proceedings of the sigchi conference on human factors in computing systems*, pages 1321–1330, 2012.

[15] L. Dinur-Klein, P. Dannon, A. Hadar, O. Rosenberg, Y. Roth, M. Kotler, and A. Zangen. Smoking cessation induced by deep repetitive transcranial magnetic stimulation of the prefrontal and insular cortices: a prospective, randomized controlled trial. *Biological psychiatry*, 76(9):742–749, 2014.

[16] H. Do, S. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

[17] 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 D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, editors, *Foundations of Software Engineering*, pages 754–766. ACM, 2021.

[18] M. Endres, Z. Karas, X. Hu, I. Kovelman, and W. Weimer. Relating reading, visualization, and coding for new programmers: A neuroimaging study. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 600–612. IEEE, 2021.

[19] 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*, 2018.

[20] 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*, pages 175–186. IEEE, 2017.

[21] D. Ford, M. Behroozi, A. Serebrenik, and C. Parnin. Beyond the code itself: how programmers really look at pull requests. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 51–60. IEEE, 2019.

[22] Z. P. Fry, B. Landau, and W. Weimer. A human study of patch maintainability. In *International Symposium on Software Testing and Analysis*, pages 177–187. ACM, 2012.

[23] L. Gazzola, D. Micucci, and L. Mariani. Automatic software repair: A survey. *IEEE Transactions on Software Engineering*, 45(1):34–67, 2017.

[24] J. H. Goldberg and J. I. Helfman. Comparing information graphics: a critical look at eye tracking. In *BEyond time and errors: novel evaLuation methods for Information Visualization*, pages 71–78, 2010.

[25] J. H. Goldberg and X. P. Kotval. Computer interface evaluation using eye movements: methods and constructs. *Journal of industrial ergonomics*, 24(6):631–645, 1999.

[26] M. Hegarty, R. E. Mayer, and C. E. Green. Comprehension of arithmetic word problems: Evidence from students' eye fixations. *Journal of educational psychology*, 84(1):76, 1992.

[27] P. B. Henderson. Mathematical reasoning in software engineering education. *Communications of the ACM*, 46(9):45–50, 2003.

[28] M. J. Hermida, M. S. Segretin, L. M. Prats, C. S. Fracchia, J. A. Colombo, and S. J. Lipina. Cognitive neuroscience, developmental psychology, and education: Interdisciplinary development of an intervention for low socioeconomic status kindergarten children. *Trends in Neuroscience and Education*, 4(1-2):15–25, 2015.

[29] B. Holdsworth and C. Woods. *Digital logic design*. Elsevier, 2002.

[30] Y. Huang, K. Leach, Z. Sharafi, N. McKay, T. Santander, and W. Weimer. Biases and differences in code review using medical imaging and eye-tracking: Genders, humans, and machines. In *Foundations of Software Engineering*, page 456–468, 2020.

[31] Y. Huang, X. Liu, R. Krueger, T. Santander, X. Hu, K. Leach, and W. Weimer. Distilling neural representations of data structure manipulation using fMRI and fNIRS. In *International Conference on Software Engineering*, pages 396–407, 2019.

[32] T.-Y. Jiang, C.-N. Liu, and J. Y. Jou. Estimating likelihood of correctness for error candidates to assist debugging faulty hdl designs. In *2005 IEEE International Symposium on Circuits and Systems*, pages 5682–5685. IEEE, 2005.

[33] J. Jung, A. Bungert, R. Bowtell, and S. R. Jackson. Vertex stimulation as a control site for transcranial magnetic stimulation: a concurrent tms/fmri study. *Brain stimulation*, 9(1):58–64, 2016.

[34] M. A. Just and P. A. Carpenter. A theory of reading: from eye fixations to comprehension. *Psychological review*, 87(4):329, 1980.

[35] R. Keim. What is a Hardware Description Language (HDL)?, 2020. Retrieved Jan 11, 2021 from `https://www.allaboutcircuits.com/technical-articles/what-is-a-hardware-description-language-hdl/`.

[36] S. Kirbas, E. Windels, O. McBello, K. Kells, M. Pagano, R. Szalanski, V. Nowack, E. R. Winter, S. Counsell, D. Bowes, T. Hall, S. Haraldsson, and J. Woodward. On the introduction of automatic program repair in Bloomberg. *IEEE Software*, 38(4):43–51, 2021.

[37] J. Kloosterman and D. Fontenot. *Comprehensive Studies Program (CSP) Support Planning Discussions AY 2019-2020*. 2020. University of Michigan meeting held on 08/04/2020.

[38] 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*, ICSE '20, page 678–690, 2020.

[39] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 3–13. IEEE, 2012.

[40] C. Le Goues, S. Forrest, and W. Weimer. Current challenges in automatic software repair. *Software quality journal*, 21(3):421–443, 2013.

[41] P. L. Li, A. J. Ko, and J. Zhu. What makes a great software engineer? In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 700–710. IEEE, 2015.

[42] K. Liu, A. Koyuncu, D. Kim, and T. F. Bissyandé. Tbar: revisiting template-based automated program repair. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 31–42, 2019.

[43] Y. Liu, L. Zhang, and Z. Zhang. A survey of test based automatic program repair. *Journal of Software*, 13(8):437–452, 2018.

[44] F. Long and M. Rinard. Automatic patch generation by learning correct code. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 298–312, 2016.

[45] M. M. Mano and M. Ciletti. *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013.

[46] M. Manoharan and B. Kaur. Mathematics teachers' perceptions of diagrams. *International Journal of Science and Mathematics Education*, pages 1–23, 2022.

[47] M. Martinez and M. Monperrus. Astor: A program repair library for java. In *Proceedings of International Symposium on Software Testing and Analysis*, 2016.

[48] S. Mechtaev, J. Yi, and A. Roychoudhury. Directfix: Looking for simple program repairs. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 448–458. IEEE, 2015.

[49] S. Mechtaev, J. Yi, and A. Roychoudhury. Angelix: Scalable multiline program patch synthesis via symbolic analysis. In *Proceedings of the 38th international conference on software engineering*, pages 691–701, 2016.

[50] E. Mendelson. *Introduction to mathematical logic*. CRC press, 2009.

[51] J. Mock, S. Huber, E. Klein, and K. Moeller. Insights into numerical cognition: Considering eye-fixations in number processing and arithmetic. *Psychological Research*, 80(3):334–359, 2016.

[52] M. Monperrus. The living review on automated program repair. Technical Report hal-01956501, HAL/archives-ouvertes.fr, 2018.

[53] J. A. Naglieri and D. Johnson. Effectiveness of a cognitive strategy intervention in improving arithmetic computation based on the pass theory. *Journal of learning disabilities*, 33(6):591–597, 2000.

[54] R. Netzel, B. Ohlhausen, K. Kurzhals, R. Woods, M. Burch, and D. Weiskopf. User performance and reading strategies for metro maps: An eye tracking study. *Spatial Cognition & Computation*, 17(1-2):39–64, 2017.

[55] H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra. Semfix: Program repair via semantic analysis. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 772–781. IEEE, 2013.

[56] J. Parkinson and Q. Cutts. The effect of a spatial skills training course in introductory computing. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 439–445, 2020.

[57] D. L. Parnas. Predicate logic for software engineering. *IEEE Transactions on Software Engineering*, 19(9):856–862, 1993.

[58] D. L. Parnas. Software engineering programs are not computer science programs. *IEEE software*, 16(6):19–30, 1999.

[59] R. D. Pea and D. M. Kurland. On the cognitive prerequisite of learning computer programming (tech. rep.). *New York: Bank Street College of Education, Center of Children and Technology.(ERIC Document Reproduction Service No ED 249 931)*, 1983.

[60] B. Peischl and F. Wotawa. Automated source-level error localization in hardware designs. *IEEE Design & Test of Computers*, 23(1):8–19, 2006.

[61] J. H. Perkins, S. Kim, S. Larsen, S. P. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. T. Sullivan, W. Wong, Y. Zibin, M. D. Ernst, and M. C. Rinard. Automatically patching errors in deployed software. In J. N. Matthews and T. E. Anderson, editors, *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009*, pages 87–102. ACM, 2009.

[62] Y. Qi, X. Mao, Y. Lei, Z. Dai, and C. Wang. The strength of random search on automated program repair. In *Proceedings of the 36th International Conference on Software Engineering*, pages 254–265, 2014.

[63] Z. Qi, F. Long, S. Achour, and M. Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 24–36, 2015.

[64] W. V. O. Quine. *Philosophy of logic*. Harvard University Press, 1986.

[65] J.-C. Ran, Y.-Y. Chang, and C.-H. Lin. An efficient mechanism for debugging RTL description. In *The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, 2003. Proceedings.*, pages 370–373. IEEE, 2003.

[66] K. Rayner. Eye movements in reading and information processing: 20 years of research. *Psychological bulletin*, 124(3):372, 1998.

[67] J. R. Reisel, M. Jablonski, H. Hosseini, and E. Munson. Assessment of factors impacting success for incoming college engineering students in a summer bridge program. *Intl. J. of Mathematical Education in Sci. and Tech.*, 43(4):421–433, 2012.

[68] K. H. Rosen. *Discrete mathematics and its applications*. McGraw-Hill, 7th edition, 2012.

[69] B. Rush and A. Ogborne. Program logic models: expanding their role and structure for program planning and evaluation. *Canadian Journal of Program Evaluation*, 6(2):95–106, 1991.

[70] S. Saha et al. Harnessing evolution for multi-hunk program repair. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 13–24. IEEE, 2019.

[71] S. Salehi, S. Cotner, and C. J. Ballen. Variation in incoming academic preparation: Consequences for minority and first-generation students. In *Frontiers in Education*, volume 5, page 552364. Frontiers Media SA, 2020.

[72] P. Santiesteban, Y. Huang, W. Weimer, and H. Ahmad. Cirfix: Automated hardware repair and its real-world applications. *IEEE Transactions on Software Engineering*, 2023.

[73] K. U. Sarker, A. B. Deraman, and R. Hasan. Descriptive logic for software engineering ontology: Aspect software quality control. In *2018 4th International Conference on Computer and Information Sciences (ICCOINS)*, pages 1–5. IEEE, 2018.

[74] P. Schober, C. Boer, and L. A. Schwarte. Correlation coefficients: appropriate use and interpretation. *Anesthesia & analgesia*, 126(5):1763–1768, 2018.

[75] E. Schulte, Z. P. Fry, E. Fast, W. Weimer, and S. Forrest. Software mutational robustness. *Genetic Programming and Evolvable Machines*, 15(3):281–312, 2014.

[76] Z. Sharafi, T. Shaffer, B. Sharif, and Y.-G. Guéhéneuc. Eye-tracking metrics in software engineering. In *Asia-Pacific Software Engineering Conference*, pages 96–103, 2015.

[77] Z. Sharafi, B. Sharif, Y.-G. Guéhéneuc, A. Begel, R. Bednarik, and M. Crosby. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25(5):3128–3174, 2020.

[78] Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc. A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology*, 67:79–107, 2015.

[79] H. H. Shen. Chinese l2 literacy development: Cognitive characteristics, learning strategies, and pedagogical interventions. *Language and Linguistics Compass*, 7(7):371–387, 2013.

[80] 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 *Proceedings of the 36th International Conference on Software Engineering*, pages 378–389, 2014.

[81] 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 *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 140–150, 2017.

[82] A. I. Sonmez, D. D. Camsari, A. L. Nandakumar, J. L. V. Voort, S. Kung, C. P. Lewis, and P. E. Croarkin. Accelerated tms for depression: a systematic review and meta-analysis. *Psychiatry research*, 273:770–781, 2019.

[83] S. Sorby, N. Veurink, and S. Streiner. Does spatial skills instruction improve stem outcomes? the answer is 'yes'. *Learning and Individual Differences*, 67:209–222, 2018.

[84] S. A. Sorby, E. Nevin, A. Behan, E. Mageean, and S. Sheridan. Spatial skills as predictors of success in first-year engineering. In *IEEE Frontiers in Education Conference*, pages 1–7, 2014.

[85] S. Staber, B. Jobstmann, and R. Bloem. Finding and fixing faults. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 35–49. Springer, 2005.

[86] C. A. Stanich, M. A. Pelch, E. J. Theobald, and S. Freeman. A new approach to supplementary instruction narrows achievement and affect gaps for underrepresented minorities, first-generation students, and women. *Chemistry Education Research and Practice*, 19(3):846–866, 2018.

[87] A. Susac, A. Bubic, M. Planinic, M. Movre, and M. Palmovic. Role of diagrams in problem solving: An evaluation of eye-tracking parameters as a measure of visual attention. *Physical Review Physics Education Research*, 15(1):013101, 2019.

[88] Y. Terao and Y. Ugawa. Basic mechanisms of tms. *Journal of clinical neurophysiology*, 19(4):322–343, 2002.

[89] J. Terrell, A. Kofink, J. Middleton, C. Rainear, E. Murphy-Hill, C. Parnin, and J. Stallings. Gender differences and bias in open source: Pull request acceptance of women versus men. *PeerJ Computer Science*, 3:e111, 2017.

[90] P. A. Tolley, C. M. Blat, C. R. McDaniel, D. B. Blackmon, and D. C. Royster. Enhancing the mathematics skills of students enrolled in introductory engineering courses: Eliminating the gap in incoming academic preparation. *Journal of STEM Education: Innovations and Research*, 13(3), 2012.

[91] A. P. Trevizol, P. Shiozawa, I. A. Cook, I. A. Sato, C. B. Kaku, F. B. Guimarães, P. Sachdev, S. Sarkhel, and Q. Cordeiro. Transcranial magnetic stimulation for obsessive-compulsive disorder: an updated systematic review and meta-analysis. *The journal of electroconvulsive therapy (ECT)*, 32(4):262–266, 2016.

[92] K. Yim, D. D. Garcia, and S. Ahn. Computer science illustrated: Engaging visual aids for computer science education. In *Proceedings of the 41st ACM technical symposium on computer science education*, pages 465–469, 2010.

[93] D. Yoon and N. H. Narayanan. Mental imagery in problem solving: An eye tracking study. In *Proceedings of the 2004 symposium on Eye tracking research & applications*, pages 77–84, 2004.