Circuit Troubleshooting Techniques in an Electrical and Computer Engineering Laboratory

Mr. Michael Kinsel, University of Virginia

Electrical Engineering Student at the University of Virginia

Caroline Elizabeth Crockett, University of Virginia

Caroline Crockett is an assistant professor at the University of Virginia in the Electrical and Computer Engineering department. She received her PhD degree from the University of Michigan in electrical engineering. Her research interests include image processing and conceptual understanding.

Dr. Natasha Smith, University of Virginia

Dr. Smith is a Professor at the University of Virginia

Dr. George Prpich, University of Virginia

Professional Skills and Safety are my main pedagogical interests. I use the Chemical Engineering laboratory to implement safety training to improve safety culture, and to adapt assessment methods to enhance development of students' professional skills. I am an Assistant Professor of Chemical Engineering at the University of Virginia and I hold a B.Sc. (University of Saskatchewan) and Ph.D. in Chemical Engineering (Queen's University). Complimenting my pedagogical research is an interest in bioprocess engineering, environmental engineering, environmental risk management, and I have authored >40 peer reviewed publications in these fields. I'm also active in developing workforce development initiatives, specifically within the biopharmaceutical manufacturing space. Beyond academia, I have 7+ years of international consulting experience working with the U.K. government, European Union, and the United Nations.

Circuit Troubleshooting Techniques in an Electrical and Computer Engineering Laboratory

Abstract

This research investigates what troubleshooting methods undergraduate electrical and computer engineering students employ when working with breadboarded circuits. While the literature in computer science clearly lays out many debugging strategies for coding, there are few equivalents in electrical and computer engineering (ECE) for hardware troubleshooting strategies. Thus, the purpose of this research is to identify troubleshooting methods in ECE, with the goal of helping educators evaluate and eventually improve students' troubleshooting ability in an engineering laboratory. This qualitative, observational study documents the approaches sophomore-level ECE students use while troubleshooting a circuit with faults intentionally added. The circuit contained resistors, operational amplifiers, power supplies, and a diode. To succeed, students had to understand the system, test the circuit, locate the faults, fix the faults, and then assess the fixes. Overall, 41% of students fixed all the faults. The most commonly used troubleshooting strategies were tracing, full system testing, gaining domain knowledge, and pattern matching. The most uncommon strategies were analytical reasoning and rebuilding.

1 Introduction

Lab instructors generally agree that troubleshooting is a vital skill for student success [1] because mistakes are inevitable and part of the learning process. Although instructors typically agree troubleshooting is important, it is often not an explicit course learning goal and it is not taught as a separate skill beyond the mechanics of using a specific troubleshooting tool [2], [3]. Understanding the skills and methods that students employ while troubleshooting, along with the obstacles they confront, can provide helpful insights for troubleshooting instruction going forward.

Michaeli and Romeike [4] define troubleshooting as "the process of locating the reason for a system malfunction and the subsequent repair or replacement of the faulty component." Thus, troubleshooting requires students to draw on broader skills such as gathering sensory information, forming predictive models, using appropriate instrumentation, and analyzing data [5]. Troubleshooting is broadly relevant in a variety of domains and professional contexts [6]. This paper is primarily concerned with two domains: electrical and computer engineering (ECE) and computer science (CS). To differentiate the two, we use "troubleshooting" to refer to identifying and resolving physical issues within electrical circuits, while we use "debugging" to refer to detecting and fixing logical errors in software code to ensure proper program functionality. Further, we use "faults" to refer to hardware errors in circuitry and "bugs" to refer to logic errors in software.

One common place that ECE students first encounter troubleshooting in their major courses is in a circuits or electronics lab course. Working with circuits can be difficult for ECE students because it requires many different types of knowledge [6] as well as the simultaneous application

of new skills with still-developing intuition. For example, students must understand the operation and expected output of the intended circuit, have general domain knowledge of each component, know common faults and constraints, and be able to use troubleshooting approaches and equipment. Most novices' awareness and knowledge of these things are weak, making troubleshooting challenging and frustrating for many. Estrada and Atwood [7] found difficulties with equipment and troubleshooting were the most common source of student frustration in physics laboratories (over twice as common as students reported being frustrated by theoretical concepts or confusing lab documents). In another study, Burkholder *et al.* [8], found significant gaps between experts' and novices' abilities to employ predictive frameworks when problem-solving.

There are many more studies on debugging code than troubleshooting ECE circuits. These CS studies categorize common debugging strategies, which is helpful in both research and instructional contexts. **The purpose of this paper is to identify and illuminate novice circuit troubleshooting strategies**. We focus on novice strategies to discover opportunities for helping students develop more mature attitudes and techniques. Thus, the purpose of this work is to aid troubleshooting assessment with the ultimate goal of improving troubleshooting instruction.

In the following sections, we provide background on processes and strategies defined in the literature before presenting our study of a troubleshooting activity given to second-year students in a circuits lab. This methodology includes the development of a codebook to characterize student strategies and an analysis of how these strategies correlate to success. We conclude with a summary of our findings and a vision for future work.

2 Background

Novice and expert troubleshooters exhibit distinct approaches rooted in their varying levels of experience and expertise within the technical domain. Novice troubleshooters are characterized by their minimal experience and their tendency to possess a limited understanding of system workings [9]. Their conceptual models are often less refined, reducing their ability to effectively diagnose complex issues [6]. Novices typically engage in broad, surface-level assessments of problems before diving into detailed examinations of specific system faults [10]. Overall, their troubleshooting approach lacks the systematic and comprehensive perspective seen in experts, resulting in a less structured and at times erratic problem-solving process [11].

In contrast, expert troubleshooters possess a wealth of domain-specific knowledge and extensive experience. Their conceptual models are well-developed and finely tuned, enabling them to quickly grasp the nuances of system behavior and potential fault states [6]. Experts adopt a systematic and holistic approach when troubleshooting, beginning with a broad assessment of the problem and a thorough comparison of error scenarios [6], [10]. They then proceed to conduct in-depth investigations into the system's function and structure, enabling them to formulate well-informed hypotheses about the root causes of errors and devise effective solutions.

Overall, the disparity between novice and expert is largely due to the extent to which they take a systematic approach to their problem-solving processes, which enables them to use their domain knowledge efficiently to diagnose and resolve technical issues.

Structured frameworks have been devised to aid novice individuals in troubleshooting, a skill of-

ten not explicitly taught in educational settings. While many possess an innate understanding of troubleshooting from activities such as household repairs, frameworks prove valuable for novices, assisting them in developing systematic problem-solving abilities [12]. One widely recognized troubleshooting framework, applicable regardless of domain expertise, condenses the process into five steps (often performed iteratively) [9]:

1. **System comprehension**
2. **Testing**
3. **Locating error localization**
4. **Error rectification**
5. **Error evaluation**

This structured approach to troubleshooting, with its general applicability, proves advantageous in training novices, especially in the absence of practical experience. Research has demonstrated that it enhances both the quantity and quality of problems resolved [13]. Analogous to algorithms or structured processes, this explicit instructional framework steers novice troubleshooters through a sequence that experts instinctively navigate with less conscious effort.

Novices tend to struggle most with the first three troubleshooting steps [9], and particularly in locating the error, or "finding the problem." Many troubleshooting strategies focus on completing these first few steps; some the key strategies that transcend domains include [6]:

- **Trial and Error:** Novices often employ this method, systematically trying different solutions until the problem is resolved.
- **Exhaustive:** This strategy involves thorough examination and testing of all possible causes to pinpoint the issue.
- **Topographic:** This strategy focuses on understanding the system's structure and components to trace the source of the problem.
- **Split-Half:** To use this strategy, you divide the system into parts and isolate the problem by testing each section separately. This is often referred to as chunking.
- **Discrepancy Detection:** This strategy emphasizes the identification of discrepancies or deviations from expected system behavior to locate faults.

Of these strategies, researchers have observed that novice programmers tend to resort to trial and error when debugging software programs [13]. Understanding and mastering additional, more systematic troubleshooting strategies could aid novices in improving their troubleshooting abilities in various domains.

3 Methodology

To identify circuit troubleshooting techniques, we conducted an exploratory, qualitative study. The study involved the following primary steps:

1. Based on the background research and student responses to a homework problem, we developed an *a priori* codebook of possible ECE troubleshooting strategies.

2. Based on feedback from teaching assistants and student work, we refined the codebook.
3. We designed a circuit with four faults of varying difficulties, with the goal of eliciting a range of troubleshooting strategies.
4. Using think-aloud interviews, we observed the circuit troubleshooting process of n=53 ECE students, each of whom were individually asked to find and repair the faults in a circuit to achieve an expected output waveform.
5. We coded all troubleshooting strategies and quantized the observational data for analysis.

The following sections describe each of these steps in more detail.

3.1 Initial Codebook Design

Coding qualitative data consists of assigning a label to a data "chunk" to capture some underlying meaning. Coding is a form of data analysis, not simply data preparation, *i.e.*, the act of coding helps the researcher to uncover themes and trends in their data [14]. The codebook design was critical to achieving both good data collection and analysis because we did not record the think-aloud interviews. Instead, the observers captured students' troubleshooting strategies in real-time. While the observers included free-hand notes in their field notes, they relied on the categories outlined in the codebook as a shorthand that allowed them to keep pace with the students.

We started the codebook development process by drawing parallels with the well-established corpus of computer science literature. This literature review allowed us to hypothesize that the following seven debugging strategies from the field of computer science were transferable to ECE [13], [15], [16].

Gaining Domain Knowledge: In the computer science context, gaining domain knowledge involves gathering the required information to make an educated attempt at solving a programming problem or fixing a bug. In an ECE context, this strategy involves interpreting schematics, datasheets, or expected results to have the knowledge needed to make an educated attempt to fix a fault in circuitry.

Tracing: Developers commonly use debugging techniques such as code tracing or error tracking to look into the intricacies of code and find the root cause of errors, *e.g.*, by adding print statements throughout the code, stepping through with a debugger, or by manually following along with the code execution on paper [13], [16]. Hardware troubleshooting may parallel this concept, with engineers using tools like oscilloscopes and multimeters to analyze electrical signals along a signal flow path to identify the source of malfunctions in circuits or systems.

Testing (System Level): In both CS and ECE, system-level tests may take the form of verifying the expected output for sample input values, sometimes as given by the specifications for the system [13]. In CS, the output is often verified using print statements, while in a circuits laboratory, it may involve taking measurements of a waveform on an oscilloscope.

Isolation: Computer scientists can isolate specific modules or functions to identify the source of errors by selectively commenting out code, utilizing debugging tools, or conducting unit tests [13].

Electrical and computer engineers adopt a similar approach by physically isolating sections of a circuit to observe the impact on overall system behavior, aiming to pinpoint the faulty element.

Pattern Matching: Computer scientists leverage pattern recognition to identify common issues, such as missing braces, or recurring problems in code, facilitating a more efficient resolution process [13]. Similarly, electrical and computer engineers may apply pattern matching to recognize common failure modes in electrical circuits, such as an op amp operating too close to its power supplies or a diode installed backward.

Considering Alternatives: In both the computer science [13] and ECE context, considering alternatives involves evaluating different possible sources of a bug or fault.

Understanding: In a computer science setting, Murphy *et al.* [13] noted that all students read the code to understand what it did, though only one explicitly noted this was their goal. In a circuit laboratory, understanding could take the form of reviewing the physical circuit, reading datasheets, and examining schematics to understand the purpose of each component before testing.

3.2 Refining the Codebook

We piloted and iteratively refined the codebook and the faulty circuit through practice think-aloud interviews with undergraduate ECE Teaching Assistants (TAs). This allowed us to garner additional real-world data, and incorporate in vivo codes, thus achieving a more comprehensive representation of troubleshooting strategies. Another benefit of the pilot think-aloud interviews was practicing using the codebook in real-time.

We also refined the codebook by studying student responses elicited from the following homework prompt given in the studied course:

> Imagine your group member asks for help troubleshooting their circuit because the gain and cutoff frequency are not what they expect. After quickly looking over all the connections, you do not see any obvious errors in the wiring. Besides checking all component values, name three troubleshooting steps you would suggest taking to try to figure out what the error is.

Based on the think-aloud pilot interviews and the student homework responses, we introduced the following additional codes.

Random & Directed Tinkering: Murphy *et al.* [13] defines tinkering as a student making "fairly random and usually unproductive changes" to software in the debugging process. While refining the codebook with teaching assistants, we found two variations of this strategy emerge. *Random tinkering* describes when a student does not know the source of the fault, so starts making arbitrary changes with the hope of solving the problem through luck. *Directed tinkering* refers to when a student locates a fault but does not know how to fix it, so makes arbitrary changes within a specific subsystem with the hopes of fixing the fault.

Analytical: While refining the codebook, some of the TAs noted that the faulty circuit lent itself to analytical analysis, particularly surrounding the nonlinear characteristics of the diode in the circuit and the op amp's gain resistors.

Rebuilding: While refining the codebook, one TA decided to clear the breadboard of its components and to rebuild the circuit from scratch. Although students could similarly recode an algorithm from scratch, we had not seen this in the CS debugging literature.

The TA and student feedback ultimately rendered the codebook clearer and more contextually relevant, as it allowed us to make the definitions and descriptions of troubleshooting techniques more accurate and reflective of student actions. Tab. 1 summarizes the final codebook.

3.3 Exercise Design

Following Van De Bogart *et al.* [17], we designed an authentic troubleshooting exercise with preset faults of varying difficulty. We started with a circuit similar to that used in [17], but added components and faults in an attempt to elicit a wider range of troubleshooting strategies.

Fig. 1 depicts the correct circuit diagram. This circuit can be split into three main parts. The first stage is an operational amplifier (op amp) configured as a voltage follower, meaning the output should exactly track the input within the limits of operation. When functioning properly, this stage does nothing to the input signal, but the voltage follower is an important design component as it acts as a buffer by providing a near-infinite input resistance and near-zero output resistance. The second stage is a diode, which we generally model as rectifying the input signal[1].

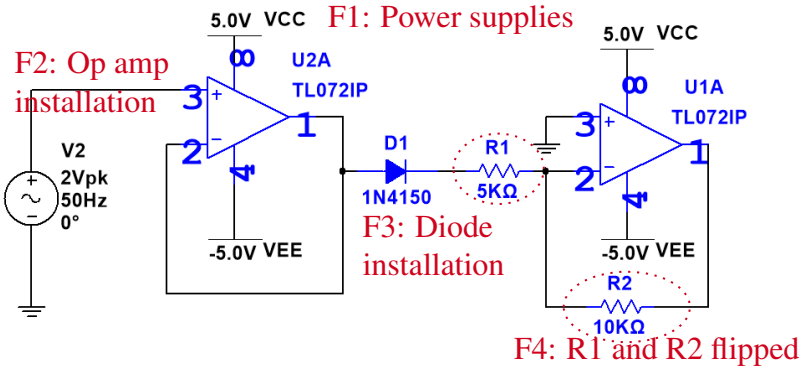

Figure 1: Schematic of the (correct) circuit with the four faults labeled.

Finally, the third stage is an inverting op amp stage with a gain of $g = \frac{-R_2}{R_1}$. When operating properly, the output of an inverting op amp is proportional to the input of the op amp with proportionality constant g. Students have built multiple circuits with each of these sub-blocks and we expect them to recognize their standard operation immediately.

We included four faults in the circuit, ordered here by location in the circuit:

(F1) **Power supplies turned off.** The power supply functionality was not enabled in the Waveforms software. To fix this, participants must click the run button on the supplies screen.

[1]Rectification involves setting any negative values to 0 and thus only keeping only the positive portion of an input signal. A more accurate (but still approximate) model is the threshold model–the diode is "off" for any input less than an approximately 0.7V threshold and drops 0.7V for any larger input, resulting in an input-output relationship described as $y(t) = \max(x(t) - 0.7, 0)$ where $y(t)$ is the output and $x(t)$ is the input.

However, until (F2) is fixed, the protection circuitry in the AD2 will automatically turn off the power supply.

(F2) **First op amp installed upside-down.** The first op amp (U2A in Fig. 1) was installed reverse from the conventional direction, meaning that pins 5-8 were in the locations of where pins 1-4 should be and vice versa. This creates an overcurrent condition, which

Code	**Definition**	**Example Actions or Quotes**
Tracing	Reviewing the flow of voltage and current in the circuit, referencing the schematic	• Visually inspecting components • Visually inspecting wires
Testing (system-level)	Using an oscilloscope to view an output signal	• Placing an oscilloscope probe on output voltage node
Gain Domain Knowledge	Reviewing specifications, reexamining the desired output, or reviewing analytical derivations to gain insight	• Reading schematics • Reading expected outputs
Pattern Matching	"Fixing" something that does not look "right", such as adjusting a component because the reference designator is misoriented, or cross referencing between experimental and expected outputs	• "This op amp looks flipped." • "I don't think the diode is supposed to look like this"
Understanding	Looking through the schematic to figure out what each stage of the circuit is supposed to do	• "This is some kind of half wave rectifier because of the bias of the diode." • "This is definitely an inverting amplifier"
Directed Tinkering	Making educated, but still indiscriminate changes to a portion of the circuit. Often involves taking action 'on a hunch'	• "I think the incorrect output has something to do with the resistors. I am going to flip them to see what happens"
Considering Alternatives	Considering possible reasons why a specific issue is occurring	• "I don't get why my output is distorted. I wonder if it's the wiring."
Random Tinkering	Making arbitrary actions towards trying to solve the circuit. Often involves taking an action "just because" or "to see what happens"	• "I'm going to flip this diode to see what happens"
Isolation	Isolating one or more systems of the circuit to test its function	• Observing the output of the diode when the diode is connected to the op amp
Using Tools	Using external tools to make adjustments or make measurements	• Adjusting oscilloscope range • Using multimeter
Rebuilding	Taking apart the whole circuit or a section of the circuit in an attempt to rebuild it correctly	• Completely taking apart the inverting amplifier in an attempt to rebuild it
Analytical	Explicitly using mathematical derivations or reasoning to find discrepancies	• Doing scratchwork on paper

Table 1: Codebook of troubleshooting strategies, listed from most to least commonly used among the sample student population.

automatically turns off the power supply. The orientation of the chip is denoted by a small circle near pin 1.

(F3) **Diode installed backward.** The diode (D1 in Fig. 1) was installed with the cathode facing the first op amp rather than the second. This fault will result in negative voltages/currents passing from the first half of the circuit rather than positive voltages/currents. Participants had to rotate the diode to correct this fault.

(F4) **Gain resistors flipped.** The gain resistors (R1 and R2 in Fig. 1) were switched, leading to a gain of 0.5 rather than 2 for the inverting op amp stage. This is similar to the first fault analyzed in [17].

3.4 Course Context and Data Collection

This study considered students in ECE 2066: Fundamentals of ECE II in spring 2023 at the University of Virginia (UVA). The three-course fundamentals ("FUN") sequence covers materials traditionally taught in circuits, electronics, and signals and systems courses. All three courses, abbreviated FUN 1, FUN 2, and FUN 3, are four credit, six contact hour, studio-style courses, with students completing labs in groups during class time using the same Analog Discovery 2 and the corresponding Waveforms software [18] that we used in this research. Thus, students have many hours of experience working with the specific hardware set-up used in this study. The FUN 2 course is required for ECE majors and minors and the majority of the students are in their second-year of study. Ref. [19] further details the ECE fundamentals curriculum.

The troubleshooting exercise was given as an in-class assignment at the end of the semester, with students receiving full credit for attempting the exercise. Over the course of three course meetings, during which students were otherwise working on their semester projects, two of the course TAs pulled students out individually to attempt the exercise. To explain the exercise, the TAs read the following script:

> Today you are going to be doing a circuit troubleshooting exercise. Please do not open the folder or uncover the circuit until time starts. Given the components on the board, you will be tasked to fix this circuit to achieve an output that matches the ones given in the folder. You will not need any additional components to complete this task. Throughout this exercise, as you troubleshoot this circuit, please speak your thought process out loud. As the exercise continues, you may be reminded to continue to speak aloud. The time limit is 10 minutes. Lastly, I will likely not be able to communicate during this exercise, so questions will generally not be answered. Thank you.

Each student was then handed the circuit schematic and expected output and given 10 minutes to try to find and correct the faults.

Following standard think-aloud protocols [20], the TAs minimized interaction during the exercise and only requested participants to continue to talk aloud if they remained silent for an extended period of time. While the participant worked on the circuit, the TAs recorded their actions according to the codebook in Tab. 1. The TAs additionally noted which specific component or sub-circuit the student was working on at any given time and any strategies that did not fit the codebook.

Of the 89 students enrolled in FUN 2, n=53 students agreed to participate in the study. There was no incentive to participate in the research. All recruitment procedures and interaction was approved by the UVA institutional review board.

Data analysis primarily involved quantitative analysis of the frequency of each code. After transcribing and collecting data from handwritten notes into a spreadsheet, we used a python script to analyze code frequency and the relationship between a particular troubleshooting strategy and whether the student was successful in finding the faults in the circuit.

4 Results

Considering each of the four faults in the circuit, 83% of students fixed (F1) power supplies turned off, 91% of students fixed (F2) op amp installed upside down, 75% of students fixed (F3) diode installed backward, and 58% of students fixed (F4) gain resistors flipped. In total, 41% (22/53) students successfully fixed all four faults in the provided time.

The most commonly used strategies by both successful and unsuccessful students were testing, tracing, pattern matching, and gaining domain knowledge. Analytical troubleshooting was the least used strategy, with only one student using this strategy. Represented in Fig. 2, successful students had slightly higher usage rates of strategies like testing, tracing, and pattern matching, while having lower or almost equal usage rates for every other strategy. Fig. 2 shows that more students employed tinkering strategies, both random and directed, instead of performing any component isolation strategies; students would often do things "on a hunch" then test the final output to "see what happens," rather than complete systematic intermediate steps.

We found that many students, both who succeeded and failed the exercise, faced significant difficulties in recognizing the hardware issues on the board. This is most apparently reflected by the strategy of random tinkering, as this strategy had the highest failure rate of 85% (only 3 of the 20 Students who used this strategy were successful at completing the circuit). Surprisingly, students who spent time employing the strategy of trying to "understand" the circuit had a failure rate of 71% (31 students used this strategy, and 9 were successful). These subsections of students glaringly represent the shortcomings of students in finding and identifying sources of faults. Fitzgerald *et al.* [9] corroborates this finding, which found that students consistently named "finding the problem" as the more difficult troubleshooting stage (more than understanding the code, testing, and fixing the problem) and that for many students, the difficulty of troubleshooting is not in repairing the error, but rather understanding the system, testing the system, or locating the error.

Fig. 2 shows the results regarding the strategy of "understanding" somewhat juxtapose what computer science debugging literature proposes, showing that taking time to understand the relevant system is not a very effective troubleshooting technique, as most of the students who tried to fully understand the function of the circuit failed to complete the exercise. The 10-minute time limit of the exercise likely also contributed to this discrepancy. Some students were able to understand what each part of the circuit is supposed to do when analyzing the schematic. However, translating understanding from the schematic to physical hardware fixes may not be as seamless as it may be for the methodological equivalent in computer science.

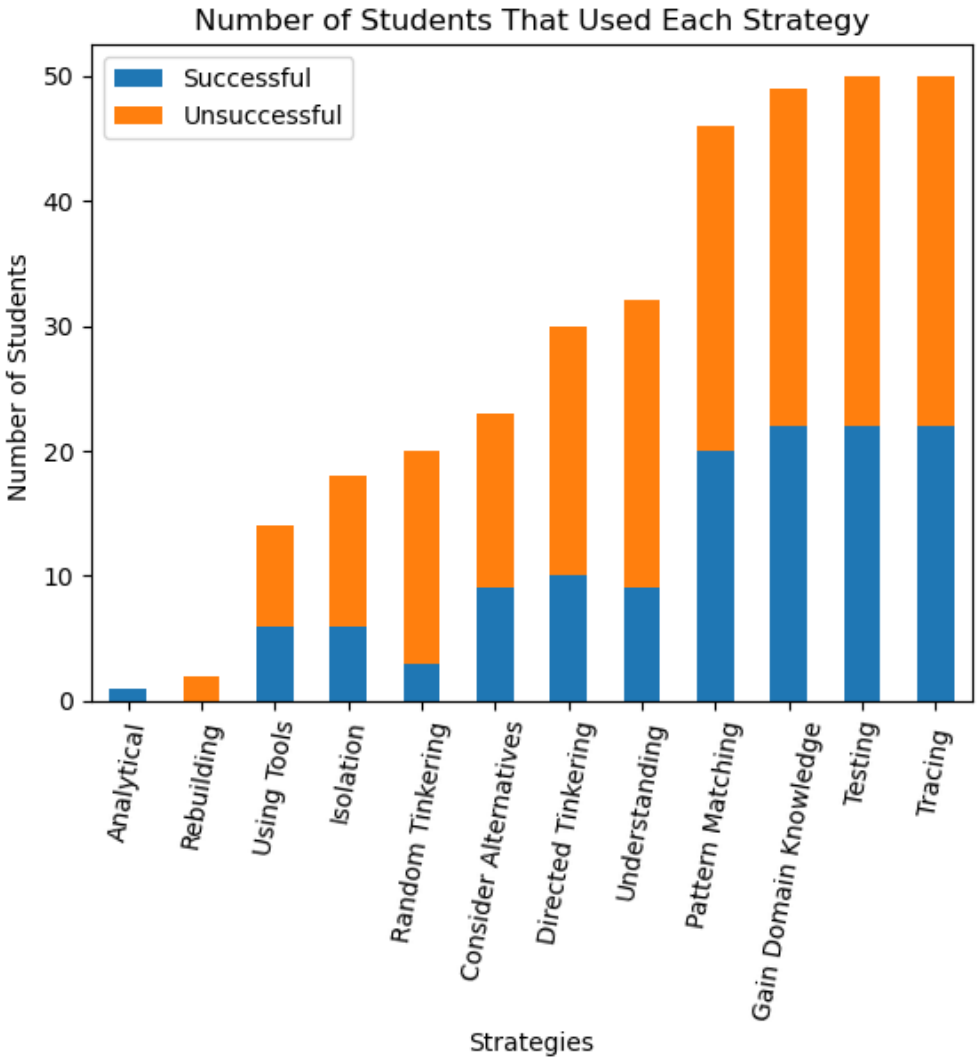

Figure 2: Troubleshooting strategy use for all students (*n=53*)

5 Discussion

Our findings offer insights into methodologies students employ when troubleshooting electronic circuits, directly addressing the main research question (What are the strategies undergraduate ECE students employ to troubleshoot hardware?) of this work. In our study, students used 12 unique troubleshooting strategies to fix faults in a circuit, most of which paralleled debugging strategies outlined in Fitzgerald *et al.* [9] and Murphy *et al.* [13].

We found that a majority of the students used testing, tracing, gaining domain knowledge, and pattern matching as primary strategies during the troubleshooting process. Conversely, there were very few students who used an analytical approach to troubleshooting, rebuilding the circuit, or using tools to troubleshoot the circuit. The strategies of isolation, random tinkering, direct tinkering, understanding, and considering alternatives were moderately used strategies. There was no clear strategy that was a marker for success, but strategies such as rebuilding and random tinkering

were observable markers for failure, as less than 20% of the students who used those strategies were successful at completing the circuit. Further, students generally found it more difficult to find the faults within the circuit than to take action to fix the faults. This would often lead students to engage in strategies that were not conducive to success, such as random tinkering or considering alternatives. Students also did not use the strategy of isolation to find faults in the circuit, with only 34% of students (18/53) employing this strategy during the exercise. These findings corroborate the findings of Fitzgerald *et al.* [9] and Michaeli and Romeike [4], bringing consistency and credence to our findings. For example, [9] also observed the use of pattern matching, isolating the problem, and tracing as debugging trends among novice computer science students.

Unit testing in computer science debugging literature is one of the predominant strategies to fix bugs in a computer program. A majority of students either use or acknowledge this debugging approach [9], [13], [16]. The troubleshooting equivalent to unit testing would be the strategy of isolation in the context of this work. We found that for troubleshooting hardware, isolating components was not a commonly used strategy by students. Out of 53 students in this study, only 34% (18/53) students tried to use isolation to locate and fix faults in the circuit. Instead, students continually used pattern matching and full system testing on the circuit, without isolating intermediate portions of the circuit.

6 Future Work

Beyond the results of this study, one of the notable achievements of this work is the development of the codebook and validation of the process for conducting troubleshooting experiments. We intend to expand the research with additional experiments to elicit whether aspects of the study design, *e.g.*, types of faults introduced, time constraints, teaming vs. individual performance, or pedagogical interventions, affect the troubleshooting techniques used. In addition, we would like to study upper-level students and experts to identify how techniques vary with expertise. For example, in the programming context, Fitzgerald *et al.* [9] found that experts are more likely to take a breadth-first approach rather than a depth-first approach when debugging and are more likely to try to understand the code before trying to correct it. Meanwhile, Jonassen and Hung [6] identifies the distinguishing feature of expert troubleshooting as the ability to generate rich mental models of the system. It would be interesting to see whether similar differences exist for troubleshooting hardware.

The ultimate goal for this line of research is to determine effective ways to improve troubleshooting ability. Current suggestions from the literature focus on helping students recognize troubleshooting as a distinct skill. Fitzgerald *et al.* [9] suggests explicitly teaching it as a skill by, for example, having students stop and consider alternatives, having students practice identifying issues in outputs, and encouraging meta-cognition about the process so students see when they are not making progress. Another suggestion is to encourage more student autonomy by emphasizing how to find the problem, not what the problem is [3]. Other ideas include: combining procedural training (*i.e.*, step-by-step guidance) with instruction system structure and concepts, using simulations and/or intelligent tutoring systems, and explicitly teaching troubleshooting strategies [6], [21]. In addition to implementing appropriate pedagogical interventions, a natural next step is to explicitly assess students based on their demonstrated troubleshooting skill.

7 Conclusion

Dounas-Frazer and Lewandowski [1] showed that developing students' ability to troubleshoot problems and emphasizing the expectation that students should expect that nothing will work the first time is essential to learning and being 'useful' in the lab. In this study, 53 ECE students attempted to correct a circuit with a series of intentional faults to determine the troubleshooting strategies they would use. Their activities were recorded and coded based on a taxonomy of strategies gleaned from prior work in the context of computer science. The study found a few differences in the strategies between successful and unsuccessful students, namely that unsuccessful students were more likely to use random tinkering or to rebuild the circuit from scratch. However, both groups widely used strategies such as tracing, testing, gaining domain knowledge, and pattern matching.

The codebook included in this paper draws parallels between computer program debugging strategies and circuit troubleshooting strategies. For researchers involved in the complexities of hardware development, this codebook can be a structured and systematic guide to troubleshooting. In instructional contexts, such as classrooms or online learning environments, this codebook can be a resource for novice students as a practical guide to hardware troubleshooting. Moreover, the codebook allows for enhanced reproducibility and collaboration in future work.

Having a list of local troubleshooting strategies that are specific to the discipline can aid instructors in explicitly teaching and assessing troubleshooting skills. This first study concentrated on novices to see what strategies they are already using as these may be easiest to incorporate in low-level courses. Future work is planned to assess the impact of pedagogical interventions on improving students' troubleshooting skills and to study how their approaches change as they grow in both domain knowledge and practical experience.

8 Acknowledgements

Thank you to the students who participated in this study, as well as to Adam Dirting for helping with data collection.

References

[1] D. R. Dounas-Frazer and H. J. Lewandowski, "Nothing works the first time: An expert experimental physics epistemology," en, in *2016 Physics Education Research Conference Proceedings*, Sacramento, CA: American Association of Physics Teachers, Dec. 2016, pp. 100–103. DOI: 10.1119/perc.2016.pr.020.

[2] B. Siegmund, M. Perscheid, M. Taeumel, and R. Hirschfeld, "Studying the Advancement in Debugging Practice of Professional Software Developers," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, Naples, Italy: IEEE, Nov. 2014, pp. 269–274, ISBN: 978-1-4799-7377-4. DOI: 10.1109/ISSREW.2014.36.

[3] T. Michaeli and R. Romeike, "Current Status and Perspectives of Debugging in the K12 Classroom: A Qualitative Study," en, in *2019 IEEE Global Engineering Education Conference (EDUCON)*, Dubai, United Arab Emirates: IEEE, Apr. 2019, pp. 1030–1038, ISBN: 978-1-5386-9506-7. DOI: 10.1109/EDUCON.2019.8725282.

[4] T. Michaeli and R. Romeike, "Investigating Students' Preexisting Debugging Traits: A Real World Escape Room Study," en, in *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, Koli Finland: ACM, Nov. 2020, pp. 1–10, ISBN: 978-1-4503-8921-1. DOI: 10.1145/3428029.3428044.

[5] L. D. Feisel and A. J. Rosa, "The Role of the Laboratory in Undergraduate Engineering Education," en, *Journal of Engineering Education*, vol. 94, no. 1, pp. 121–130, Jan. 2005, ISSN: 10694730. DOI: 10.1002/j.2168-9830.2005.tb00833.x.

[6] D. H. Jonassen and W. Hung, "Learning to Troubleshoot: A New Theory-Based Design Architecture," en, *Educational Psychology Review*, vol. 18, no. 1, pp. 77–114, Mar. 2006, ISSN: 1040-726X, 1573-336X. DOI: 10.1007/s10648-006-9001-8.

[7] T. Estrada and S. Atwood, "Factors that Affect Student Frustration Level in Introductory Laboratory Experiences," en, in *2012 ASEE Annual Conference & Exposition Proceedings*, San Antonio, Texas: ASEE Conferences, Jun. 2012, pp. 25.629.1–25.629.7. DOI: 10.18260/1-2--21386.

[8] E. Burkholder, A. M. Price, M. Flynn, and C. E. Wieman, "Assessing problem-solving in science and engineering programs," in *2019 Physics Education Research Conference Proceedings*, Provo, UT: American Association of Physics Teachers, Jan. 2020. DOI: 10.1119/perc.2019.pr.Burkholder.

[9] S. Fitzgerald, R. McCauley, B. Hanks, L. Murphy, B. Simon, and C. Zander, "Debugging From the Student Perspective," en, *IEEE Transactions on Education*, vol. 53, no. 3, pp. 390–396, Aug. 2010, ISSN: 0018-9359, 1557-9638. DOI: 10.1109/TE.2009.2025266.

[10] I. Vessey, "Expertise in Debugging Computer Programs: An Analysis of the Content of Verbal Protocols," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 5, pp. 621–637, Sep. 1986, ISSN: 0018-9472. DOI: 10.1109/TSMC.1986.289308.

[11] C. Crockett, G. Prpich, and N. Smith, "Experimental Self-Efficacy and Troubleshooting Ability in a Chemical Engineering Laboratory," en, in *2023 ASEE Annual Conference*, Jun. 2023. [Online]. Available: https://peer.asee.org/43573.

[12] A. Schaafstal, J. M. Schraagen, and M. Van Berl, "Cognitive Task Analysis and Innovation of Training: The Case of Structured Troubleshooting," en, *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 42, no. 1, pp. 75–86, Mar. 2000, ISSN: 0018-7208, 1547-8181. DOI: 10.1518/001872000779656570.

[13] L. Murphy, G. Lewandowski, R. McCauley, B. Simon, L. Thomas, and C. Zander, "Debugging: The good, the bad, and the quirky – a qualitative analysis of novices' strategies," en, in *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, Portland OR USA: ACM, Mar. 2008, pp. 163–167, ISBN: 9781595937995. DOI: 10.1145/1352135.1352191.

[14] M. B. Miles, A. M. Huberman, and J. Saldaña, "Chapter 4: Fundamentals of qualitative data analysis," English, in *Qualitative data analysis : a methods sourcebook*, Thousand Oaks, CA: SAGE Publications, 2014, pp. 69–104, ISBN: 978-1-4522-5787-7.

[15] R. McCauley, S. Fitzgerald, G. Lewandowski, *et al.*, "Debugging: A review of the literature from an educational perspective," en, *Computer Science Education*, vol. 18, no. 2, pp. 67–92, Jun. 2008, ISSN: 0899-3408, 1744-5175. DOI: 10.1080/08993400802114581.

[16] T. D. LaToza, M. Arab, D. Loksa, and A. J. Ko, "Explicit programming strategies," en, *Empirical Software Engineering*, vol. 25, no. 4, pp. 2416–2449, Jul. 2020, ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-020-09810-1.

[17] K. L. Van De Bogart, D. R. Dounas-Frazer, H. J. Lewandowski, and M. R. Stetzer, "Investigating the role of socially mediated metacognition during collaborative troubleshooting of electric circuits," en, *Physical Review Physics Education Research*, vol. 13, no. 2, p. 020 116, Sep. 2017, ISSN: 2469-9896. DOI: 10.1103/PhysRevPhysEducRes.13.020116.

[18] *Analog discovery 2*. [Online]. Available: https://digilent.com/shop/analog-discovery-2-100ms-s-usb-oscilloscope-logic-analyzer-and-variable-power-supply/.

[19] H. C. Powell, R. W. Williams, M. Brandt-Pearce, and R. Weikle, "Restructuring an electrical and computer engineering curriculum: A vertically integrated laboratory/lecture approach," en, Gainesville, Florida: ASEE, Apr. 2015. [Online]. Available: http://se.asee.org/proceedings/ASEE2015/papers2015/53.pdf.

[20] K. Anders Ericsson and Herbert A. Simon, *Protocol Analysis: Verbal Reports As Data*, English. Cambridge, Mass: A Bradford Book, 1993, vol. Rev. ed, ISBN: 978-0-262-55023-9.

[21] C. Li, E. Chan, P. Denny, A. Luxton-Reilly, and E. Tempero, "Towards a Framework for Teaching Debugging," en, in *Proceedings of the Twenty-First Australasian Computing Education Conference*, Sydney NSW Australia: ACM, Jan. 2019, pp. 79–86, ISBN: 978-1-4503-6622-9. DOI: 10.1145/3286960.3286970.