Investigating Students' Preexisting Debugging Traits: A Real World Escape Room Study

Tilman Michaeli
Computing Education Research Group
Friedrich-Alexander-Universität Erlangen-Nürnberg
91058 Erlangen, Germany
tilman.michaeli@fau.de

Ralf Romeike
Computing Education Research Group
Freie Universität Berlin
14195 Berlin, Germany
ralf.romeike@fu-berlin.de

ABSTRACT

Being able to find and fix errors is an essential skill in computer programming. Nevertheless, debugging poses a major hurdle in the K12 classroom, as students are often rather helpless and rely on the teacher hurrying from one student-PC to the other. Overall, there is a lack of respective concepts and materials for the classroom as well as research on how to teach debugging. According to the constructivist learning theory, teaching and developing concepts and materials for the classroom must take learners' preexisting experience into account to be effective. In their daily lives, students are confronted with errors long before they build programming experience: Whether there is a problem with "the internet" or with their bicycle, they are *troubleshooting* and locating and fixing errors. Debugging is a special case of general troubleshooting and shares common characteristics, such as the overall process or particular strategies. Thus, the aim of this study is to investigate students' preexisting debugging traits. To this end, we developed a real-world escape room consisting of debugging-related troubleshooting exercises. This allows us to observe students' troubleshooting behavior in a natural environment. Building upon this, we employed the escape room approach with around 150 high school students and analyzed the resulting video data. Based on the data we identify preexisting debugging traits such as students struggling to generate hypotheses or to undo changes. Furthermore, they are not able to effectively test a system and struggle with cognitive load in topographic search. Therefore, our study firstly contributes to understanding and explaining the behavior of novice debuggers. The second contribution is an innovative methodology to analyze preexisting debugging traits.

CCS CONCEPTS

• **Social and professional topics → K-12 education**.

KEYWORDS

debugging, escape room, computational thinking, troubleshooting, computer science education, K12

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Koli Calling '20, November 19–22, 2020, Koli, Finland
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8921-1/20/11…$15.00
https://doi.org/10.1145/3428029.3428044

ACM Reference Format:
Tilman Michaeli and Ralf Romeike. 2020. Investigating Students' Preexisting Debugging Traits: A Real World Escape Room Study. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research (Koli Calling '20), November 19–22, 2020, Koli, Finland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3428029.3428044

1 INTRODUCTION

In programming, finding and fixing errors is an important skill for both experts and novices. While professional developers spend between 20% and 40% of their working time on debugging code [38], debugging poses a major hurdle for students learning to program [23]. Furthermore, debugging is one of the approaches to Computational Thinking [46] and hence considered important for every student.

If we look into K-12 classrooms, we see that teachers lack adequate concepts and materials for fostering and addressing debugging in the classroom [30]. Most of the time is spent rushing from one student to another, trying to help the students in their struggles, an approach best described as "putting out the fires". Explicit teaching of a debugging process, certain debugging strategies, or the use of debugging tools rarely takes place in the classroom [30]. In consequence, novices are often left alone with their errors and forced to learn debugging "the hard way" – the same way most professionals report they have "learned" debugging [38].

According to the learning theory of constructivism, learning is a constant and active process of refining preexisting models of a subject by making and reflecting on new experiences [39]. Therefore, we have to incorporate learners' preexisting experience to eventually develop suitable approaches, best practices, and materials for the classroom [2]. This is also emphasised in the concept of educational reconstruction [7]. Therefore, the aim of this study is to investigate preexisting debugging traits of K-12 students.

But what preexisting experience on debugging is there? Despite being considered an aspect of programming, debugging is already part of students' daily lives in the form of troubleshooting: Debugging is a special case of troubleshooting – troubleshooting in the domain of programming [19]. Therefore, we developed debugging-related troubleshooting exercises and conducted a study among K-12 students. By analyzing students' troubleshooting processes and strategies we can identify certain preexisting traits that influence or explain novices' debugging behavior and need to be addressed in teaching.

In contrast to existing approaches (cf. [41]) that used real world examples of debugging situations such as giving a description of what to do when a light bulb stops working, we take this approach a step further: Instead of asking participants for how they would react if they *were put into a given situation*, we actually put them in that situation by using a real world escape room setting.

2 RELATED WORK

2.1 Preexisting Debugging Experience

There is a large amount of related work on the investigation of learners' preexisting experience for various subjects. For the domain of programming, Onorato and Schvanveldt [35] as well as Miller [31] investigated preconceptions by studying learners "programming" in natural language. Gibson and O'Kelly [12] analysed students' problem solving process for search problems, Kolikant [22] investigated students' preconceptions regarding concurrency and synchronisation, and in the commonsense computing series [24, 42, 43], preconceptions for various topics such as sorting, concurrency or logic were investigated.

What kind of preexisting experience on debugging do students have? They frequently "debug" in their daily lives before they learn to program: If, for example, their bicycle or "the internet" stops working, they start to troubleshoot. Troubleshooting is the process of locating the reason for a system malfunction and the subsequent repair or replacement of the faulty component [32]. Debugging is troubleshooting in the domain of programming, a special case of general problem solving [17, 19]. Therefore, similar skills and strategies are involved in troubleshooting and debugging, as [25] points out.

The study conducted by Simon et al. [41] within the commonsense computing series is central to this paper. They investigated preexisting debugging experiences for university students by analyzing their troubleshooting behavior to conclude implications for teaching debugging. To this end, they asked the participants for their reactions in four real world troubleshooting situations, such as giving instructions to repair a broken light bulb, troubleshooting the popular childrens' game "telephone", or describing their reaction for further real world troubleshooting instances from the participant's life. The subjects, university students, answered in written form. The authors then analyzed those answers for common characteristics and compared them to the debugging behaviors of novices and of experts. From the results, they conclude implications for teaching debugging such as the need to "address the differences between locating an error and fixing it", emphasizing the importance of test-only, or that undoing a previous step is an unnatural behaviour for students. They conclude that debugging is far less "common sense" than sorting or concurrency is.

Thus, similar to Simon et al. [41], we consider real-life troubleshooting experience to be preexisting debugging experience that has to be kept in mind when teaching debugging. However, in contrast to topics like concurrency, we consider this experience to be not quite preconceptions, but rather traits: instead of concepts and attitudes towards debugging, we look at actual skills and behavioral patterns.

Looking into troubleshooting - and debugging as a special case of it - outlines the similarities between the two practises: The debugging process, a high-level systematic pursuit of a plan to find and correct errors is characterized as follows (see for example [19]): First, the program is tested and the failure observed. Then, if necessary, an overview of the program is sought. In the next step, hypotheses are formulated and experimentally verified. If necessary, the hypothesis is refined repeatedly. Eventually, the error is corrected and the program is tested once more. Zeller [47] labels this approach the "scientific method". Since debugging is troubleshooting in the domain of programming, the debugging process can be seen as a specialization of the general troubleshooting process described by Schaafstall et al. [40] (see figure 1).

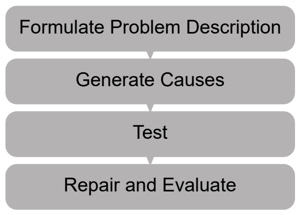


Figure 1: Visualization of the troubleshooting process based on Schaafstal et al. [40]

The same applies to debugging strategies. These are lower-level practices that support the steps of finding and refining hypotheses. Examples for this are tracing the program flow using print-debugging or the debugger, slicing and commenting out or forcing the execution of a specific case (see for example [29] or [38]). Jonassen and Hung [18] consider these strategies to be local troubleshooting strategies, applicable only to a specific domain, such as debugging. For the debugging strategy of tracing, the global counterpart (and therefore general troubleshooting strategy which is independent of context) is a *topographic search*, which can go either forward or backward through the program. For the local strategy of forcing the execution of a specific case and comparing the actual program output to the expected output, *functional/discrepancy detection* can be seen as the respective global – context-independent – strategy, and so on [25].

However, on the topic of transferring skills (such as computer science-related approaches and concepts according to Computational Thinking), results have often been underwhelming [14]. In contrast, indications for the transfer of debugging skills beyond the domain of programming have been shown to exist in a study by Carver and Risinger [4]. They gave students one hour of debugging training as part of a larger Logo curriculum. It contained a flowchart characterizing the debugging process, bug mappings and debugging "diaries" that were always present in the classroom. Besides an improvement in students' code debugging skills, Carver and Risinger found improved performance, such as a higher accuracy and a more focused search, in the non-computer transfer tasks.

In summary, we see that debugging is special case of troubleshooting. Therefore, the debugging process as well as debugging strategies such as tracing or testing (*local strategies*) are manifestations of the general troubleshooting process or *global troubleshooting strategies* such as *topographic search* or *functional/discrepancy detection* which are context-independent. Furthermore, existing research suggests, that preexisting debugging experience in the form of troubleshooting influences debugging behavior. Nevertheless, there is not enough insight into preexisting debugging traits of high school students - and how they might influence their debugging behavior such as the overall debugging process or certain debugging strategies.

2.2 Escape Room Games as Research Approach

There is an increasing number of studies regarding the usage of real world escape rooms in various areas of both informal and formal educational settings. Using an escape room as a method for education provides various pedagogical opportunities, such as an engaging and motivating context for learning by applying elements of game-based learning (e.g. [3, 34]). Furthermore, skills such as collaboration, critical thinking, and problem solving can be fostered in a natural way [11, 15, 36]. The content ranges from exercises and riddles that involve mostly general problem solving or team coordination skills (e.g. [11, 45]) to knowledge specific to a certain domain or subject, such as computer science [3], pharmacy [8], physics [44], and many more.

In many educational escape rooms, design criteria, student motivation and learning success are analyzed. The potential of using escape rooms as a research method for analyzing students learning or problem solving processes, however, remains predominantly untapped.

Järveläinen and Paavilainen-Mäntymäki [16] conducted a comparative case study in which they analyzed the learning processes of three student teams in the context of a research method in information systems science. They found that the different teams employed different learning processes in their paths throughout the escape room.

In the field of computer science education, Hacke [15] analyzed behavioral patterns in problem solving processes for their educational computer science escape room. To this end, they investigated the video recordings of 38 groups. For their analysis, they used a deductive category system to classify behavioral patterns in the problem solving process. Afterward, the influence of those patterns regarding the overall success in the game was evaluated. They found promising behavioral patterns such as using the whiteboard, having a coordinator rather than a leader for the group behaviour, or having structured task solvers in the team. The majority of those patterns are on a rather abstract "organisational" or "team composition and characteristics" level, and do not focus on the actual problem solving processes in greater detail.

In summary, we see that applying escape room scenarios as a research method is a – thus far – overlooked, but promising method. It enables us to observe and study problem-solving processes in an organic environment. This way, we can take studying students' preexisting debugging traits one step further than has been done so far: Instead of only letting participants describe how they would react to and proceed in a given situation, we can observe the actual troubleshooting processes and behavior in a real troubleshooting situation. Therefore, employing troubleshooting exercises related to debugging within an escape room scenario enables us to identify preexisting debugging traits better and more easily.

3 METHODOLOGY

The aim of this study is to identify students' preexisting debugging traits. This helps us understand and explain traits of novice debuggers, thus providing the basis for developing concepts and materials on teaching debugging for the classroom. As discussed above, students' preexisting debugging experience consists of troubleshooting situations in their daily lives. Therefore, we investigate students' behavior in different debugging-related troubleshooting situations. This enables us to identify certain preexisting debugging traits by studying students' troubleshooting process and strategies.

Therefore, we will address the following research question:

RQ Which preexisting debugging traits can we observe within students?

3.1 Study Design

To answer the research question, we applied a cross-sectional study design. We developed a live-action escape room scenario consisting of different debugging-related troubleshooting exercises. By analyzing high-school students' overall troubleshooting process in debugging-related scenarios and how they apply certain troubleshooting strategies, we can identify common *preexisting debugging traits*.

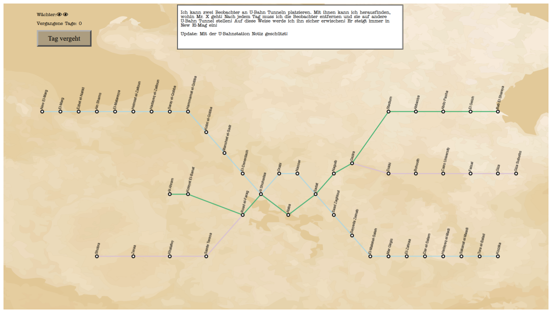
The escape game itself followed the typical rules: The participants had one hour to solve all the puzzles and riddles, open a variety of locks, and win the game (our game was set in ancient Egypt: the goal was to find a stolen object to end the Pharaoh's curse). Via a surveillance camera mounted in the room, the participants' behavior and audio could be recorded. We used the camera's two-way audio feature to record the participants' communication and to interact with them to control the game if needed.

Similar to the study Simon et al. [41] conducted with university students, we used troubleshooting exercises to determine the participants' preexisting experience. Instead of having the participants describe their course of action in writing, the escape room setting provides the following advantages:

- We observe the actual troubleshooting process and strategies in a natural environment. Furthermore, the participants are not able to extensively plan or revise their "final answer" as they would be in an assessment in written form.
- We assess students' reactions if their initial plan does not work out.
- Teamwork and open communication between team members makes the troubleshooting process observable, as e.g. Fields et al. point out [9].

3.2 Subjects

The escape room was employed on seven different occasions. Each time, one school class visited our university to play the room. Overall, about 150 students in groups of four to six played the room. In total, we had 28 teams attending the escape room. All students were from upper German high-schools and 14 to 18 years old. Some of the students had computer science experience, but at most a year of formal computer science education. Furthermore, some had previous experience with escape room games, which might provide advantages with certain "meta-factors" such as how to search a room, map certain locks to certain exercises based on the solution's number of inputs, and so on. Overall, we collected 32 hours of video material.



Ich kann zwei Beobachter an U-Bahn Tunneln platzieren. Mit ihnen kann ich herausfinden, wohin Mr. X geht! Nach jedem Tag muss ich die Beobachter entfernen und sie auf andere U-Bahn Tunnel stellen! Auf diese Weise werde ich ihn sicher erwischen! Er steigt immer in New El-Mag ein!

Update: Mit der U-Bahnstation Notiz geschützt!

Figure 2: Finding Mr. X

3.3 Escape Room Exercises

The exercises for the escape room were developed in a design-based-research-process. We oriented the design according to Jonassen and Hung's [18] characterization of troubleshooting exercises as appearing ill-defined, requiring the construction of a conceptional model of the system, having known solutions with clear success criteria and usually containing only singular fault states. Furthermore, we designed the exercises in such a way that debugging-related troubleshooting behaviors such as a systematic troubleshooting process or particular troubleshooting strategies were necessary and observable. The respective troubleshooting behaviors we expected for each of the exercises are described within the results. We almost completely excluded search-and-find-tasks, which are common in escape games, as they provide no insights into our research interest. Throughout the study and the different data collections, we rotated some of the exercises in and out to adjust the difficulty of the escape game. In the following, we report only on the most successful exercises with regards to our research interest.

Screen: It is the students' task to get the screen to work. To this end, they simply have to plug in its power supply. Furthermore, a Chromecast is connected to the screen which is used for streaming the content, but marked with a “do not touch”-sticker. This activity was developed for analyzing the overall troubleshooting process and the application of a functional/discrepancy detection strategy.

Tangle of cables: The participants are confronted with a set of eight daisy-chained USB-cables connecting a LED-flashlight to a power adapter. Furthermore, they find a box which is prepared in such a way that they can not look inside without the flashlight. Two of the cables are defective. The task is to identify the broken cables and build a sufficiently long chain with the remaining wires to shine light into the box and read the numbers inside. This activity was developed for analyzing the overall troubleshooting process and the application of a functional/discrepancy detection strategy.

Tap the telephone: The participants find a box with five cables hanging out. Upon inserting a cable into one of the five sockets, they get audio feedback on how many cables are plugged into the correct socket. They have to find the right order of cables, somewhat similar to the game “mastermind”. This activity was developed for analyzing the overall troubleshooting process.

Valley of the kings: The participants find a map of the valley of the kings with a route on it. Furthermore, they find a route description, although some of the arrows describing the route are wrong. The paper with the route contains the hint to identify the 6 errors. This activity was developed for analyzing the application of a topographic search strategy.

Finding Mr. X:. A web app (after a tablet has been found) shows a map of Cairo's metro system. The participants are tasked with finding out where Mr. X is going, given his starting station. To this end, they can place two watchers on the underground tunnels and get feedback on whether Mr X passed them. After 30 seconds of waiting, they get another chance to place the watchers. This is one example of how we actively fostered communication between the team members by introducing a pause after getting feedback in the design of the exercises. This activity was developed for analyzing the application of a topographic search strategy.

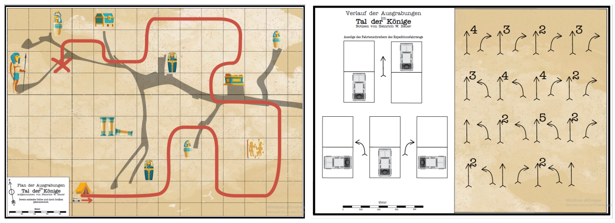


Figure 3: Valley of the kings

3.4 Data Analysis

As the aim of our study is to identify preexisting debugging traits, we need to identify certain patterns in the students' application of troubleshooting process and strategies. In order for behavioral patterns to be considered relevant traits in our study, a certain frequency of occurrence is required. Therefore, we applied an integrated approach of qualitative and quantitative analysis according to Mayring [27], consisting of three steps:

First, we analyzed the video data using a structured content analysis approach according to Mayring [28], similar to [15], to capture all student behaviors and actions for each of the exercises. To this end, we deductively developed a category system for which table 1 provides selected extracts. The category system comprises all possible behaviors the students could show and employ within the respective exercises. To avoid neglecting important aspects due to previously-defined categories, inductive additions were allowed. The recorded videos form the basis of the evaluation, while a video segment with a minimum length of 5 seconds serves as a unit of analysis. To ensure intercoder reliability, a second researcher coded the video data for six of the groups (about 20 percent of the video data).

In the next step, we arranged the categories according to the frequency of their appearance in the material. We only considered actions that were coded for at least 20 percent of the groups relevant for further analysis.

In the final step, we identified preexisting debugging traits. To this end, we interpreted the relevant actions for each individual group. This was done in a qualitative process: Depending on the

Table 1: Excerpt of the category system for structured content analysis

Exercise	Category
Tangle of cables	plug daisy-chain into socket
	check individual cables
	extend/shorten chain step by step
	randomly extend/shorten chain
	check only the flashlight
	check cable connections
Screen	finding the unplugged cable
	pressing the screens power button
	pressing other buttons
	examining the screen
	checking the screens connections
	checking the sockets connections
	checking the area around the screen
	plugging in the cable

context, in which the action was exhibited, we mapped it either to a step of the troubleshooting process (see figure 1), or to a particular troubleshooting strategy. After mapping, we then analyzed its respective characteristics (e.g. how they approached a certain step of the troubleshooting process or applied a particular strategy). The generalization of such characteristics over all groups form the preexisting debugging traits.

4 RESULTS

In the following section, we will describe the observed troubleshooting behavior and strategies for the individual tasks. For every exercise, we will first outline our hypotheses regarding the students' approaches according to the troubleshooting process (see figure 1) and strategies. Subsequently, the actual behavior observed will be described. In conclusion, the respective results are summed up and interpreted with regards to preexisting debugging traits.

4.1 Screen

The screen exercise always posed the first puzzle of the room, as it provided the students with a timer and necessary information for the next exercise upon completion. We expected the students to:

(1) notice that the monitor is not working *(formulate problem description)*.
(2) hypothesize, based on this observation, that no power source is connected *(generate causes)*.
(3) press the power or other buttons, and/or check the inputs systematically *(test)*.
(4) search for the respective cable (if not spotted beforehand) and plug it in, solving the exercise *(repair and evaluate)*.

Upon analyzing the video data, we observed the following behaviors: Before entering the room, the students received the hint that "the evil professor was doing something suspicious on the screen", so all groups examined the monitor within the first minutes and noticed it not working. Some groups (30 %) immediately spotted the loose cable (lying directly beside the monitor), plugged it in and therefore solved the exercise. Other groups (about 35 %) systematically checked the connections at the screen and/or sockets and then either immediately found the cable or started searching for it. Few groups explicitly formulated a corresponding hypothesis; nevertheless, they checked a common cause for such a problem in reaction to the *internal* hypothesis "the screen might not be connected correctly", simply based on their everyday life experience.

A considerable percentage of the other groups (35 %) needed significantly longer amounts of time or even a hint on how to solve this exercise. Most of these groups pressed the power button, noting the lack of any reaction on the screen. Nevertheless, they did not generate any hypothesis as to the reason for this. Instead, they checked the area around the monitor in a unsystematic manner or had a look at the power sockets, but without tracing which cables go where. Some students even held the unplugged cable in their hands, but did not decide what to do with it. Eventually, they started to work on searching the room for different exercises.

Another common behavior we observed among all groups was trying to press other buttons on the screen, for example to change the input for the screen. A likely reason for this is that having chosen the wrong source is a common cause of error the students might have encountered in their daily lives. Without a power supply, they saw no feedback on the screen, so this heuristic did not help them in this particular case (besides giving more clues suggesting that no power is connected).

In summary, the data for this exercise indicates two preexisting debugging traits. First of all, some students applied a systematic troubleshooting process and incorporated experience from their everyday lives according to our expectations. However, a notable number of students was not able to generate an initial hypothesis as to why the screen was not working. Overall, we were surprised by the problems a lot of students had with this task, as well as the degree of helplessness shown, which is common in debugging as well [37].

4.2 Tangle of Cables

For the tangle-of-cables-exercise, we expected the students to:

(1) plug in the daisy-chained USB cables, noticing the flashlight not working *(formulate problem description)*.
(2) generate multiple hypotheses based on this observation, such as the power socket, the power adapter, the cables, or the flashlight itself not working *(generate causes)*.
(3) systematically check those hypotheses by testing the respective component *(test* by using strategies such as *functional/discrepancy detection*).
(4) eventually conclude that some of the cables are not working, identify those cables and build the daisy-chain without them, allowing them to shine light into the box and read the numbers inside *(repair and evaluate)*.

As expected, all students initially tried to plug the daisy-chained USB cables with the flashlight at its end into a socket. After noticing that the flashlight does not light up, almost all groups checked the connection of the individual USB cables. This can be described as "applying common heuristics", as participants likely had made similar experiences in their everyday lives. At this point, many students explicitly hypothesized that the socket might be faulty.

To verify this hypothesis, they connected the cables to a different socket in the room. In reaction to this, the majority of groups struggled for one of two reasons:

- One part (about 37 %) of the students was **unable to generate any alternative hypothesis**. Some groups even checked a third socket. If their first approach did not work out – even after getting the hint that the sockets and flashlight are working fine – they lacked a direction to keep on working on the problem.
- Other groups (about 30 %) hypothesized or even directly asked the room operator whether the flashlight might be broken. However, they lacked an approach to **test this hypothesis**. Some groups eventually checked the flashlight with a shorter chain of cables.

To our surprise, only about 25 % of the groups thought to check whether the flashlight was working when directly connected to the power adapter. After being stuck, some groups showed an unsystematic trial-and-error-approach, such as switching the position of individual cables in the chain. A lot of groups needed hints such as to "check the cables", or "that socket and flashlight are fine". At the point they figured out that some of the cables might be broken, most groups systematically checked one cable after another to identify the broken ones. Other groups extended the chain of cables step-by-step in a systematic manner. No group applied a binary search for this problem. Given the small number of wires and (more) efficient method of checking each wire individually, using a binary search would have seemed inappropriate.

In summary, the data for this exercise suggests three preexisting debugging traits. Most students applied a systematic troubleshooting process according to our expectations and even incorporated experience from their daily lives for the tangle-of-cables exercise, but only for their first hypothesis. They had significant problems generating an alternative hypothesis if their first approach did not work out. To our surprise, if they actually had an alternative hypothesis, they struggled with verifying it, as they did not test the respective component in an isolated manner.

4.3 Tap the telephone

For this exercise, we expected the students to:

(1) randomly connect cables and sockets *(formulate problem description)*.
(2) in doing so, notice a correlation between the audio feedback and the connected wires and, therefore, understand the system *(generate causes/test)*.
(3) systematically find the correct position for each cable step by step by checking each of the remaining sockets for a cable *(repair and evaluate)*.

Most students started by plugging in all of the 5 cables. Depending on the number of matching sockets, they received audio feedback in the form of beeping sounds. All groups then started to randomly switch the positions of certain plugs. This way, the number of beeping sounds either increased or decreased. If the number of beeping sounds, and therefore the number of correctly positioned plugs, decreased, we observed an interesting pattern: Many students did not reverse the changes that led to the decrease, but kept on switching cables. One group, for example, had already correctly positioned three plugs by randomly switching positions, mostly of adjacent plugs. Nevertheless, they kept on doing so, until only one beeping sound was left, and eventually removed all the plugs and turned their attention to another task for the time being. Although we cannot say this with certainty, the majority of the groups showing this behavior seemed to have grasped the concept of "the more beeping sounds the better". Nevertheless, at least for this rather exploratory process, students did not **undo** certain changes if they reduced or did not improve the result.

Most groups switched to a more systematic approach after some time: About half of the groups removed all plugs and started over by checking all 5 positions for the first plug, then the remaining four for the next, and so on. The other half identified the already-correctly-positioned plugs by removing them one after the other, and then just tested the remaining positions for the remaining plugs.

In summary, the data for this exercise shows two preexisting debugging traits. Most groups solved this exercise according to our expectations: After *understanding the system* in an exploratory manner, they applied a systematic process of checking the right position for each cable step by step. However, we identified the interesting pattern of not *undoing* changes, which students showed at the end of their exploration of the system, although they seemed to have already grasped the meaning of the number of beeping sounds. Similar behaviors are common in debugging, where novices often add additional errors by not undoing attempted but unsuccessful fixes [13].

4.4 Valley of the Kings

For the valley-of-the-kings-exercise, we expected the students to:

(1) understand the system of arrow directions and route using the given example and comparing the first few directions with the route *(formulate problem description)*.
(2) trace the route, using some form of auxiliary material to help keep track of the wrong arrows and/or the current position *(apply forward topographic search strategy)*.

To our surprise, about 75 % of the groups traced the route more than twice. For most of these groups, the first three wrong arrow directions (of six in total) did not pose a problem, but the longer the route, the more miss-identifications or confusion was observed. We identified the following reasons for this issue: Students often tracked their progress using their fingers, but commonly only on either the map or the route – despite working in a group. This led to confusion regarding the current position after some time, especially with simultaneous group discussions taking place.

Another common pattern was that one or two students worked on the route and the map, while another student wrote down the errors they identified. Doing so, they frequently encountered problems in their communication: Some groups perfectly identified all six errors, but they forgot or miscommunicated to write one of them down, or wrote it down twice. In general, many groups started noting things down only after their first trace throughout the route. Furthermore, although they, for example, were able to mark the wrong arrows or additional information directly on the map-paper, many groups chose to do this on a separate sheet of paper – possibly because they did not want to "destroy" the game materials. Some groups even transformed the path on the map into the arrow-representation and then compared it to the route-paper, but made mistakes in the process of this transformation. In general, a common pattern was that when re-tracing, the students changed previously correctly identified wrong arrows as being correct, leading to even more confusion.

In summary, we were surprised by the problems many students encountered. We expected the exercise to be rather easy, as it was inspired by unplugged debugging exercises for primary school children, albeit with a lengthened route. While *understanding the system* such as the meaning of the arrows did not pose a problem, students struggled with keeping track of their current position while tracing and/or identifying wrong arrows as correct, especially towards the end of the route. Similar problems with tracing are common for programming novices as well [26].

4.5 Finding-Mr-X

For this exercise, we expected the students to:

(1) start from the given entry subway station
(2) systematically isolate the exit station by placing watchers that provide as much information as possible regarding the route of Mr. X, such as at tunnels after transit stations *(apply forward topographic search strategy).*

For this exercise, we observed several different patterns of students behavior. A large percentage of students (about 60 %) actually employed an optimal or close-to-optimal strategy: Their approach always started from the given entry point (the final stop of the line) of Mr. X. Building upon this, they "traced" Mr. X's route through the subway system by placing the watchers on relevant subway tunnels, i.e. those that provide a lot of information, such as stations directly after transfer points. Based on the feedback they received, they isolated and identified the subway station in question step by step in a systematic manner. While waiting the 30 seconds until they could place the watchers again, one group, for example, discussed the placement of the next watchers as follows: *(A) "Let's place the watcher here, he has to pass here" (B) "Yes, exactly, he has to pass there anyway [so this does not help us]"*. Then, B pointed out a better spot that provides more information as to where Mr. X is going: *(B) "Here, we will know whether he changes from the blue to the green metro line"*. Those groups needed only 3 to 5 iterations of placing the watchers to solve the exercise.

A common pattern for less efficient groups was to employ a different kind of *topographic strategy*: Instead of "tracing" the possible route, starting from the given entry point throughout the subway system, they placed their watchers on distant branches of the subway system to find out whether Mr. X passed them. They sometimes even chose stations close to the respective final station instead of the stations directly after transit stations. This way, they needed a significantly larger number of iterations to find the right station.

Another common pattern we saw in the data was that participants began by checking the subway tunnels right next or very close to the entry station. Only after noticing that due to the 30-second waiting period, it would take a long time to solve the exercise this way, they stopped this linear step-by-step tracing.

In summary, we observed two different topographical strategies employed by the students: One part of the students traced the route from the entry station in a rather efficient manner. Other students isolated the subway station by ruling out the branches one after the other, which appeared to be rather inefficient for this particular task. In debugging, choosing suitable locations for placing *prints* or breakpoints also poses a major challenge for novices [33].

5 DISCUSSION

Overall, most of the traits we identified by observing students troubleshooting behavior resemble the behavior novices show in debugging. Therefore, we conclude that those traits must exist before novices learn to program and are therefore independent from the programming or debugging domain. We argue that in order for students to use debugging skills successfully, these traits need to be addressed explicitly in teaching debugging by conveying corresponding strategies at an abstract level. In the following, we will generalize the results from the individual exercises and interpret them with regards to programming and debugging. Furthermore, we discuss implications for teaching debugging.

5.1 Generalization and Interpretation of the Results

Preexisting debugging trait: Students struggle with generating hypotheses, in particular alternative hypotheses. Regarding the students' overall approach, the data showed that the majority of students employed a process similar to the troubleshooting process described earlier, although they did not formulate every hypothesis explicitly. Although they generally followed a plan, students had problems at particular steps of the process. For debugging, (as well as for troubleshooting, cf. [18, 32]), being able to effectively generate good and multiple hypotheses is a decisive difference between experts and novices [13, 20]. With the screen exercise, a notable part of the students encountered problems with **formulating an initial hypothesis**. In contrast to this, all students were able to generate and test an initial hypothesis for the tangle-of-cables-exercise. We suspect that the different domain knowledge needed for those exercises explains this finding: While all students have experience with operating devices using USB-cables, not all of them are likely to have experience in working with and connecting a computer screen. Furthermore, the data indicates that students have problems with **formulating alternative or more than one hypothesis**. Many were stuck after they tested their first hypothesis for the tangle-of-cables-exercise and had to reject it. Some groups even tried a third power socket. This is in line with literature, as Bereiter and Miller report that a wrong troubleshooting path, even in light of contradicting clues, is often not discarded [1]. For debugging, Murphy et al. report that students often did not recognize they were stuck and needed to change their approach [33]. They conclude that especially thinking about alternative bug causes should be incorporated in debugging instruction. The traits we found in our study might explain the students' debugging behavior and support the importance of fostering the generation of hypotheses, in particular alternative hypotheses, in teaching debugging.

When debugging code, it is a common behavior for students to employ a trial-and-error-approach, such as adding semicolons or braces, or increase or decrease loop parameters (cf. [30]). Our data showed similar patterns only after students got stuck. This indicates that students employing such an approach in debugging may be stuck and lack any direction on how to work on the problem. However, our data does not allow us to make statements on the reason for this; whether they stopped due to being unable to form a (new) hypothesis (i.e. caused by lack of domain knowledge), or due to not realizing the need for a systematically-established hypothesis, we cannot say.

Preexisting debugging trait: Students are not able to effectively test a system and test single components in an isolated manner. In the tangle-of-cables-exercise, the students hypothesized that the power socket might be defect. To test this, they used a different socket. Murphy et al. [33] as well as Fitzgerald et al. [10] found that university students commonly employed testing in debugging programs, but mostly just used sample values provided by an instructor. They seldom use specific cases such as boundary conditions. In our study, students had problems with testing their hypothesis that the flashlight might not be working. In particular, they were not able to test the component "flashlight" in an isolated system, therefore disregarding the possibility of defect wires (as was actually the case). Based on our data, we hypothesize that students have **no preexisting experience regarding effective testing** from their everyday lives, which might contribute to their deficits in testing for debugging. Therefore, respective skills have to be fostered in teaching debugging from scratch, such as how to test a single isolated component of code.

Preexisting debugging trait: Undoing changes is not intuitive. When applying a possible fix to a particular system and evaluating the results of those changes, **students rarely resorted to reverting those changes**. Once more, Murphy et al. found similar results in their qualitative study of novice strategies in debugging code, even though there is IDE support for *undo* in programming. Because of this, it is a common pattern that novices introduce new bugs in their code when debugging [13]. Our data indicates that undoing changes is no intuitive trait students have from their daily lives, and therefore needs to be addressed explicitly in teaching debugging.

Preexisting debugging trait: Students use domain knowledge and incorporate heuristics and patterns for common error causes into their troubleshooting process. Our data shows that students incorporate previous experience into their troubleshooting process, such as checking whether the USB-cables are connected properly, whether the power supply is connected, or whether the right input for the screen is selected. Applying such heuristics and patterns for common bugs is an essential debugging skill and professional developers use these patterns and heuristics, based on their experience, to "shorten" their debugging process [30]. To support learning from past errors, many professional developers keep a debugging diary or debugging log, which they use to document their debugging experience [38]. Obviously, programming novices lack the respective experience. Therefore, teachers need to support building such patterns and heuristics by employing methods such as debugging diaries (cf. [4]) or or ensuring that situations arise in which students can learn these behaviors.

Preexisting debugging trait: Students struggle with cognitive load and lack the means to effectively employ external representation to help in topographic search and tracing. In the finding-mr-x-exercise, a lot of students actually chose well-suited subway tunnels to place their watchers. However, students struggle in applying similar tracing strategies such as print-debugging in coding as they often choose inefficient locations or text to print [33] – although print-debugging is one of the most common debugging strategies [10]. We suspect that the main problem in coding is not having or not being able to construct a mental model of the program flow, similar to the map in our exercise. Nevertheless, this provides an interesting starting point for further investigation and to compare students' placement of such statements in coding and examples like our subway map.

Students also struggled when tracing the route in the valley-of-the-kings-exercise. This surprised us, as we used a simple arrow notation and concrete (rather than symbolic) tracing [6] without any an additional hurdles such as a programming language syntax. For tracing in programming, novices' overall skills have been characterized as poor [26]. Students often lack the necessary accuracy [37], are not able to raise the abstraction level and in general have problems with cognitive load [5]. We suspect a similar problem with cognitive load in our case, as the students struggled in particular with the last three incorrect arrows. Furthermore, they used rather inefficient external representations, as is reported for tracing in coding [5]. Therefore, our data suggests that this problem (pre)exists outside of the programming domain – although our exercise was arguably very closely related to programming. To tackle efficient tracing in the classroom, students need to build experience by practising tracing and how to use external representations efficiently.

Preexisting debugging trait: Students are easily frustrated. Another interesting general observation we made was that the majority of students got frustrated rather quickly. In contrast to typical school exercises, due to the problem solving character of the escape room, they did not have any "recipe" for how to solve the different exercises. Instead of persisting and struggling their way through a task, many students gave up and preferred to search for more hints in the room or work on a different exercise. Especially if the groups split in order to work on different exercises at the same time and one group had a breakthrough, all team members came (and often stayed) together to work on the exercise, for which they now had some clues. We characterize this as seeking success instead of persisting on a, for the moment, frustrating task. We see similar reports for students debugging code [21], as debugging is a process where a certain level of persistence and perseverance is needed. Our data suggests that persevering in a task is independent from programming to a certain degree, and needs to be addressed in the classroom, for example by providing strategies to get unstuck.

5.2 Comparison to literature

As described above, Simon et al. [41] identified university students' preconceptions regarding debugging with a similar approach. Instead of using video data from an escape room, they analyzed students' answers to certain scenarios. Our methodology allowed us to analyze behavior "live in action" and in greater detail. Furthermore, our high-school participants are notably younger and likely to have significantly less troubleshooting experience from their daily lives. Additionally, in contrast to rather open scenarios, our exercises had one correct solution due to the escape room setting. This is suitable for troubleshooting exercises, as most of the time, they require a particular solution [18].

Nevertheless, our results provide additional evidence for some of their results, such as undo seeming unnatural for students and that students often do not generate alternative hypotheses. However, our methodology and the exercises developed – based on the strong theoretical basis – allowed us to find further preexisting debugging traits such as the students incorporating domain knowledge and heuristics and patterns or the problems with certain strategies such as topographic search or testing.

5.3 Remarks on the Methodology

Regarding the escape room methodology we applied, we were satisfied with the ability to observe students' behavior, process and strategies. In our data, there were groups in which one student individually solved a certain exercise on his own, as well as groups which solved a certain exercise in such way that – due to the camera angle – we were not able to observe closely. In consequence, we omitted those cases from our analysis. Nevertheless, for the vast majority (and given the large number of participants) of exercises and groups, we were able to observe the students' actions, and to get additional insights from groups discussions. Furthermore, all groups showed a high level of motivation due to the scenario. As the room was not linear, the students were sometimes able to work on certain tasks in parallel. Students were always able to search the room for further clues for later exercises. This might have influenced the tendency to give up on a certain task and focus on a different exercise for the time being (although we observed few groups that split to work on tasks in parallel overall).

For the analysis of debugging-related troubleshooting for identifying preexisting debugging traits, we consider our approach to be suitable. Certainly not all troubleshooting behavior can be transferred to the domain of debugging. For that reason, some of the exercises we developed yielded no significant insights into our research question. Therefore, these exercises were omitted from this analysis. However, the exercises we *are* reporting on in this paper show a strong connection to literature, where the correlation between debugging and troubleshooting is established. In consequence, debugging-related troubleshooting behaviors according to literature such as a systematic troubleshooting process or particular global troubleshooting strategies were necessary and observable. Our findings strongly support this assumption: The preexisting debugging traits we identified in this study are reflected in novice programmers debugging behavior and display a strong overlap with typical problems of novices as discussed above. Therefore, we consider this methodology appropriate to answer our research question. Furthermore, we deem it important to consider these results when teaching debugging.

6 CONCLUSION

The aim of this study was to investigate students' preexisting debugging traits. To this end, we observed high school students' behavior in different debugging-related troubleshooting exercises in a real world escape room scenario and analyzed the video data. Overall, our study provides the following two contributions:

Firstly, we provide an innovative methodological approach to study students' troubleshooting and problem solving behavior, processes, and strategies. The main advantage in comparison to a written assessment of participants' reactions in given situations is that we can observe the actual troubleshooting process and strategies in a natural environment, including the reactions that occur if an initial approach does not work out. The communication within the groups, which was actively fostered in the design of the exercises, turned out to make the participants' processes observable.

Secondly, the escape room approach and the respective tasks allowed us to analyze preexisting debugging traits on a strong theoretical basis – which are reflected in novices debugging behavior: Our data shows that the students employ some debugging-related practises such as a systematic troubleshooting process or the incorporation of patterns and heuristics. Nevertheless, they struggle with generating hypotheses, in particular alternative hypotheses, and undoing changes. Furthermore, they are not able to effectively test a system and test single, isolated components. For tracing, our data indicates that students struggle with cognitive load and lack the means to effectively employ external representation to help in their process.

These findings contribute to the understanding of novices' debugging behavior and allow us to incorporate learners' preexisting debugging traits into the design of suitable approaches, best practices, and materials for the classroom. The analysis of the data indicates the following implications for teaching debugging:

- **Fostering a systematic debugging process:** While many students showed a systematic troubleshooting process, they switched back to to an unsystematic trial-and-error-approach when stuck or unable to generate a hypothesis for the cause of the problem. Conveying a systematic debugging process and fostering corresponding skills such as generating multiple hypotheses and undoing changes might help students get unstuck and improve their independence in debugging, as has been shown to be successful [4].
- **Support building heuristics and patterns for common bugs:** In the troubleshooting exercises, the students applied heuristics and patterns they learned through experience. For debugging, novices lack those experiences they can build upon. Therefore, students should be actively supported in acquiring respective experience, patterns and heuristics.
- **Explicitly conveying debugging strategies:** Our data indicates that efficient testing does not play a significant role in students' daily lives, which results in a lack of experience in applying testing as a strategy for troubleshooting. Furthermore, students struggled with tracing in the context of a topographic search. Therefore, corresponding debugging strategies such as print-debugging, commenting out or testing need to be addressed explicitly in teaching debugging.

In summary, this study provides deep insights into students' preexisting debugging experience. The traits we identified can give teachers an idea of what problems their students might struggle with in debugging, and therefore an understanding of factors that contribute to the "hard way" of learning debugging. This way, they are able to support their students, making this journey a little less "hard". Building upon this, in future research we want to use the escape room exercises and setting the other way around: Instead of investigating preexisting traits, we want to analyze the influence of debugging education on troubleshooting behavior. This way, similar to the study of Carver [4], we can analyse and measure the transfer of debugging skills into students daily lives in the sense of computational thinking.

REFERENCES

[1] Susan R Bereiter and Steven M Miller. 1989. A field-based study of troubleshooting in computer-controlled manufacturing systems. *IEEE transactions on Systems, Man, and Cybernetics* 19, 2 (1989), 205–219.

[2] Jeffrey Bonar and Elliot Soloway. 1985. Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human–Computer Interaction* 1, 2 (1985), 133–161.

[3] Carlos Borrego, Cristina Fernández, Ian Blanes, and Sergi Robles. 2017. Room escape at class: Escape games activities to facilitate the motivation and learning in computer science. *JOTSE* 7, 2 (2017), 162–171.

[4] McCoy Sharon Carver and Sally Clarke Risinger. 1987. Improving children's debugging skills. In *Empirical studies of programmers: Second workshop*. Ablex Publishing Corp., Norwood, NJ, USA, 147–171.

[5] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. 2017. Using tracing and sketching to solve programming problems: replicating and extending an analysis of what students draw. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. AMC, New York, NY, USA, 164–172.

[6] Françoise Détienne and Elliot Soloway. 1990. An empirically-derived control structure for the process of program understanding. *International Journal of Man-Machine Studies* 33, 3 (1990), 323–342.

[7] Reinders Duit, Harald Gropengießer, and Ulrich Kattmann. 2005. Towards science education research that is relevant for improving practice: The model of educational reconstruction. In *Developing standars in research on science education edition*, H.E. Fischer (Ed.). Taylor & Francis, London, UK, 1–10.

[8] Heidi Eukel, Jeanne Frenzel, and Dan Cernusca. 2017. Educational Gaming for Pharmacy Students–Design and Evaluation of a Diabetes-themed Escape Room. *American journal of pharmaceutical education* 81, 7 (2017), 6265.

[9] Deborah A Fields, Kristin A Searle, and Yasmin B Kafai. 2016. Deconstruction kits for learning: Students' collaborative debugging of electronic textile designs. In *Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education*. ACM, New York, NY, USA, 82–85.

[10] Sue Fitzgerald, Renée McCauley, Brian Hanks, Laurie Murphy, Beth Simon, and Carol Zander. 2009. Debugging from the student perspective. *IEEE Transactions on Education* 53, 3 (2009), 390–396.

[11] Cheri Friedrich, Hilary Teaford, Ally Taubenheim, Patrick Boland, and Brian Sick. 2019. Escaping the professional silo: an escape room implemented in an interprofessional education curriculum. *Journal of interprofessional care* 33, 5 (2019), 573–575.

[12] Paul Gibson and Jackie O'Kelly. 2005. Software engineering as a model of understanding for learning and problem solving. In *Proceedings of the first ICER*. ACM, New York, NY, USA, 87–97.

[13] Leo Gugerty and G. Olson. 1986. Debugging by skilled and novice programmers. *ACM SIGCHI Bulletin* 17, 4 (1986), 171–174.

[14] Mark Guzdial. 2015. Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics* 8, 6 (2015), 1–165.

[15] Alexander Hacke. 2019. Computer Science Problem Solving in the Escape Game "Room-X". In *Informatics in Schools. New Ideas in School Informatics*, Sergei N. Pozdniakov and Valentina Dagienė (Eds.). Springer International Publishing, Cham, 281–292.

[16] Jonna Järveläinen and Eriikka Paavilainen-Mäntymäki. 2019. Escape Room as Game-Based Learning Process: Causation-Effectuation Perspective. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*. ScholarSpace 2019, Waikoloa Village, HI, USA, 1477–1475.

[17] David H Jonassen. 2000. Toward a design theory of problem solving. *Educational technology research and development* 48, 4 (2000), 63–85.

[18] David H Jonassen and Woei Hung. 2006. Learning to troubleshoot: A new theory-based design architecture. *Educational Psychology Review* 18, 1 (2006), 77–114.

[19] Irvin R Katz and John R Anderson. 1987. Debugging: An analysis of bug-location strategies. *Human-Computer Interaction* 3, 4 (1987), 351–399.

[20] ChanMin Kim, Jiangmei Yuan, Lucas Vasconcelos, Minyoung Shin, and Roger B Hill. 2018. Debugging during block-based programming. *Instructional Science* 46, 5 (2018), 767–787.

[21] Paivi Kinnunen and Beth Simon. 2010. Experiencing Programming Assignments in CS1: The Emotional Toll. In *Proceedings of the sixth ICER (ICER '10)*. ACM, New York, NY, USA, 77–86. https://doi.org/10.1145/1839594.1839609

[22] Yifat Ben-David Kolikant. 2001. Gardeners and cinema tickets: High school students' preconceptions of concurrency. *Computer Science Education* 11, 3 (2001), 221–245.

[23] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A study of the difficulties of novice programmers. *Acm Sigcse Bulletin* 37, 3 (2005), 14–18.

[24] Gary Lewandowski, Dennis Bouvier, Robert McCartney, Kate Sanders, and Beth Simon. 2007. Commonsense computing (episode 3) concurrency and concert tickets. In *Proceedings of the third ICER*. ACM, New York, NY, USA, 133–144.

[25] Chen Li, Emily Chan, Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2019. Towards a Framework for Teaching Debugging. In *Proceedings of the Twenty-First Australasian Computing Education Conference*. ACM, New York, NY, USA, 79–86.

[26] Raymond Lister, Elizabeth S Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, et al. 2004. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin* 36, 4 (2004), 119–150.

[27] Philipp Mayring. 2001. Combination and integration of qualitative and quantitative analysis. In *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research*, Vol. 2. 1–14.

[28] Philipp Mayring. 2004. Qualitative content analysis. *A companion to qualitative research* 1 (2004), 159–176.

[29] Robert C Metzger. 2004. *Debugging by thinking: A multidisciplinary approach*. Elsevier Digital Press, Burlington, MA.

[30] Tilman Michaeli and Ralf Romeike. 2019. Current Status and Perspectives of Debugging in the K12 Classroom: A Qualitative Study. In *2019 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, Dubai, VAE, 1030–1038.

[31] Lance A Miller. 1981. Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal* 20, 2 (1981), 184–215.

[32] Nancy M Morris and William B Rouse. 1985. Review and evaluation of empirical research in troubleshooting. *Human factors* 27, 5 (1985), 503–530.

[33] Laurie Murphy, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: the good, the bad, and the quirky–a qualitative analysis of novices' strategies. In *ACM SIGCSE Bulletin*. ACM, Portland, OR, USA, 163–167.

[34] Scott Nicholson. 2018. Creating engaging escape rooms for the classroom. *Childhood Education* 94, 1 (2018), 44–49.

[35] Lisa Onorato and Roger W Schvaneveldt. 1987. Programmer-nonprogrammer differences in specifying procedures to people and computers. *Journal of Systems and Software* 7, 4 (1987), 357–369.

[36] Rui Pan, Henry Lo, and Carman Neustaedter. 2017. Collaboration, awareness, and communication in real-life escape rooms. In *Proceedings of the 2017 Conference on Designing Interactive Systems*. ACM, New York, NY, USA, 1353–1364.

[37] David N Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. 1986. Conditions of learning in novice programmers. *Journal of Educational Computing Research* 2, 1 (1986), 37–55.

[38] Michael Perscheid, Benjamin Siegmund, Marcel Taeumel, and Robert Hirschfeld. 2017. Studying the advancement in debugging practice of professional software developers. *Software Quality Journal* 25, 1 (2017), 83–110.

[39] Denis C Phillips. 1995. The good, the bad, and the ugly: The many faces of constructivism. *Educational researcher* 24, 7 (1995), 5–12.

[40] Alma Schaafstal, Jan Maarten Schraagen, and Marcel Van Berl. 2000. Cognitive task analysis and innovation of training: The case of structured troubleshooting. *Human factors* 42, 1 (2000), 75–86.

[41] Beth Simon, Dennis Bouvier, Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, and Kate Sanders. 2008. Common sense computing (episode 4): Debugging. *Computer Science Education* 18, 2 (2008), 117–133.

[42] Beth Simon, Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, and Kate Sanders. 2006. Commonsense computing: what students know before we teach (episode 1: sorting). In *Proceedings of the second ICER*. ACM, New York, NY, USA, 29–40.

[43] Tammy VanDeGrift, Dennis Bouvier, Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, and Beth Simon. 2010. Commonsense computing (episode 6) logic is harder than pie. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. ACM, New York, NY, USA, 76–85.

[44] Alpár István Vita Vörös and Zsuzsa Sárközi. 2017. Physics escape room as an educational tool. In *AIP Conference Proceedings*, Vol. 1916. AIP Publishing, 050002.

[45] Patrick Williams. 2018. Using escape room-like puzzles to teach undergraduate students effective and efficient group process skills. In *2018 IEEE Integrated STEM Education Conference (ISEC)*. IEEE, 254–257.

[46] Aman Yadav, Ninger Zhou, Chris Mayfield, Susanne Hambrusch, and John T Korb. 2011. Introducing Computational Thinking in Education Courses. In *Proceedings of the 42Nd SIGSCE*. ACM, New York, NY, USA, 465–470.

[47] Andreas Zeller. 2009. *Why programs fail: a guide to systematic debugging*. Morgan Kaufmann, Burlington, MA, USA.