Accepted Manuscript, to appear in:
Koli Calling '25: Proceedings of the 25th Koli Calling International Conference on Computing Education Research

Investigating Debugging Processes: A Scoping Review

Elena Spörer
elena.spoerer@tum.de
Technical University of Munich
Computing Education Research Group
Munich, Germany

Tilman Michaeli
tilman.michaeli@tum.de
Technical University of Munich
Computing Education Research Group
Munich, Germany

Abstract

Debugging is a necessity in programming, in both professional and educational contexts. For novices, however, debugging is often a significant challenge. Understanding what students actually do when debugging is key to addressing difficulties and developing targeted interventions. As a result, several studies investigated the students' debugging process. Nevertheless, it is unknown which aspects have been analyzed so far and which gaps still exist. To clarify the state of research, we conducted a scoping review and identified 36 papers that analyze students' debugging processes. Our review shows that the majority of the studies focused on selected parts of the process, mainly by analyzing screen recordings or videos from the classroom using qualitative, inductive methods. Moreover, most of the papers either assessed the students' debugging strategies or their performance. As a result, there is a lack of deductive analysis approaches focusing on investigating the whole debugging process. Consequently, this review provides a starting point for future analyses of debugging processes.

CCS Concepts

• **Social and professional topics** → **Computing education**; *Computer science education.*

Keywords

debugging, scoping review, process analysis, computing education

ACM Reference Format:
Elena Spörer and Tilman Michaeli. 2018. Investigating Debugging Processes: A Scoping Review. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Despite decades of research on computing education, learning to program is still challenging for students. They struggle with several different aspects, like new concepts and numerous bugs in their programs. Consequently, debugging, the systematic process of locating and solving errors in a program, is a necessity for students, which is not only unavoidable, but also often causes frustration [50].

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

Also, from a professional point of view, debugging is an essential and time-consuming process of software development. Professional developers spend up to 40% of their time debugging code. To be successful, they apply a systematic and iterative process to find and fix bugs [51]. However, this process is not intuitive for novices and must be taught explicitly [27]. As a result, several interventions have been developed to teach students a systematic debugging process in order to facilitate successful debugging [66].

For designing effective interventions and offering targeted support, it is essential to understand how learners actually engage in the debugging process. While pre-post-measurements can demonstrate progress, they fall short in revealing what happens in between the points of measurement [55]. In contrast, investigating the process itself offers valuable insights into where learners struggle, how they approach bugs, and what kind of support may help them best [42]. Moreover, process-level data can enable the design of personalized, timely feedback or even automated scaffolding [3].

In general, debugging processes have been the subject of numerous empirical studies in computing education, and a wide range of methods, data sources, and analysis approaches have been used (e.g., [1, 16, 62]). However, so far, no comprehensive synthesis exists that provides an overview of which methodologies have been used to investigate debugging processes. To address this gap, we conducted a scoping review that investigated the current body of knowledge on students' debugging processes. To this end, this review aims to provide a synthesis of the study designs, data sources, analysis methods, and aspects investigated in prior work. Furthermore, we sought to identify research gaps and thus offer perspectives for future research in the field.

2 Related Work

As most programs do not work correctly on the first attempt, debugging is a necessity, not only in professional software development, but also in computing education. Thereby, debugging refers to the systematic process of identifying and correcting bugs in a program. An early systematic review from McCauley et al. provides a comprehensive overview of educational research on debugging until 2008. The review summarizes key findings on the causes and the types of bugs in students' programs, the prerequisites for successful debugging, the differences between novices' and experts' debugging processes, and approaches for teaching and learning debugging effectively. A central conclusion of the review was the need for explicit teaching of debugging, because debugging skills differ from programming skills and are not acquired "on the go" when learning to program [42]. In the following years, debugging moved more into the focus of teaching and research, and several teaching interventions addressing on debugging have been developed. To this

end, in 2024 a systematic review from Yang et al. (2024) synthesized knowledge about interventions on debugging. The result is a summary of pedagogical approaches and intervention modalities used for teaching debugging. The review highlights characteristics for successful interventions like the design of learning settings, such as collaboration or working unplugged. Moreover, most of the approaches focus on teaching selected steps of the debugging process, like identifying fault symptoms or diagnosing the fault, while only a few interventions address non-cognitive aspects like self-efficacy [66].

2.1 The Debugging Process

From our perspective, a **debugging process** refers to an ideal-typical sequence of steps aimed at finding and fixing bugs. Considering the current state of research, also other terms like "global strategies" [34] refer to our understanding of debugging processes. In this work, however, we use the term debugging process for this global perspective on debugging.

As debugging is described as a process, numerous empirical and theoretical process models have been developed to describe successful, ideal-typical debugging processes of professional software developers. Already in the 70s, research has investigated debugging processes. For instance, Gould (1975) observed experts during debugging, which resulted in an early, gross description of an ideal-typical process model [18]. Others, such as Gilmore (1991), combined empirical findings from observations with existing debugging process models. They concluded with a model that focuses on program comprehension and the construction of a mental representation of the problem during debugging [17]. In contrast, the model from Zeller (2009) is based on observations on how to conduct experiments in the natural sciences. This so-called scientific method was mapped to the debugging process. In doing so, this model explicitly highlights hypothesis testing through experiments to find bugs [67]. Overall, regardless of the development of the ideal-typical debugging process model, all models can be summarized into the following four steps: (1) observe the failure, (2) set up a hypothesis, (3) verify the hypothesis, and (4) fix the bug and verify the solution.

Furthermore, research shows that the debugging process may vary significantly depending on whether working with own code or code written by other developers. When debugging code written by others, additional comprehension steps are required [25], which is addressed only in some of the ideal-typical debugging process models like the one from Gilmore (1991).

Besides a systematic process, debugging strategies are context-dependent key elements for successful debugging [18, 34, 43]. Those strategies are sometimes also referred to as tactics [18], or local strategies [34]. Debugging strategies are applied during one or more steps to support the debugging process. Examples for expedient strategies are setting breakpoints in the IDE to validate a hypothesis or tracing the code for a better understanding of the program.

2.2 Analyzing Processes in Educational Research

In various domains of educational research, there is a growing interest in understanding the processes of *how* students learn instead of only measuring *what* they have learned. Consequently, recent work increasingly emphasizes the analysis of processes. This provides opportunities to identify factors that support or hinder learning progress and offers valuable insights into the processes to improve teaching [3].

Rather than only analyzing static performance outcomes, research investigating processes focuses on the temporal and sequential structure of learner behavior [55]. To this end, various data types like videos, eye gazes, and log data are used to record processes [54] which can be assessed in a qualitative and quantitative manner. When analyzing learning process data, several dimensions can be investigated to provide a more comprehensive understanding of the learning process, including cognitive, meta-cognitive, social, and motivational-affective processes [3]:

Cognitive processes refer to the mental processes of students that are required for learning and problem-solving [2]. Especially, high cognitive load hampers the acquisition of new knowledge [58]. To this end, efficient cognitive processes are essential for learners to successfully integrate new information into their long-term memory.

Meta-cognitive processes involve learners' abilities to control their learning process, e.g., to monitor, to regulate, and to reflect on their learning strategies [36]. Meta-cognitive competencies become especially relevant in situations where initial approaches fail and the consideration of alternatives is necessary [21, 36]. In such cases, the reflection of failure helps to prevent similar mistakes in the future [21]. As a result, meta-cognitive competencies are essential for sustainable learning.

Social processes emerge when learners are collaborating. Interacting with peers is important, and the discourse with others can lead to an improved understanding of the learning content [12]. However, the effectiveness of collaboration is highly influenced by the group composition [53] as it affects the co-construction of knowledge [12].

Motivational-affective processes encompass learners' emotional and motivational states. Grounded in the self-determination theory from Deci and Ryan (2004), experiencing autonomy, competence, and relatedness helps the learners to develop intrinsic motivation, which supports positive emotions [9]. Moreover, intrinsic motivation is associated with more and longer engagement, as well as with higher learning outcomes [49].

In summary, process analysis offers great potential for understanding *how* students learn. This potential has also been recognized in computing education. To this end, in previous research, not only programming processes were analyzed in-depth [6, 26], but also debugging processes have been taken into account already. Nevertheless, there is still a lack of a synthesis of the existing literature on debugging processes.

3 Methodology

The aim of this paper is to synthesize previous research that investigated debugging processes in the context of computing education research. We want to provide a comprehensive overview of which analysis approaches were used to assess debugging processes, which aspects were analyzed, and what gaps are still unaddressed. To this end, we answer the following four RQs:

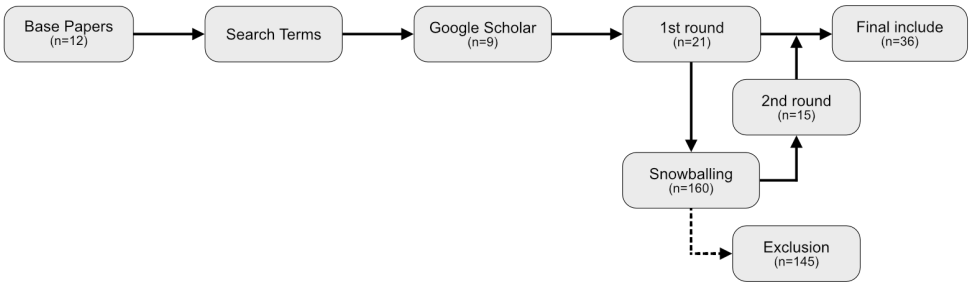

Figure 1: Visualization of the Paper Selection Process

RQ1: How was the study designed to capture debugging processes? First, we analyzed how the studies to capture debugging processes were set-up, as they are the foundation for further analysis. Due to the changes in the debugging process when debugging code written by others [25], we documented *which code* was used for the study. Moreover, especially for younger students, working unplugged is particularly useful, as they benefit from embodiment and tangible experiences [66]. As unplugged activities also support code comprehension, we evaluated whether the study was conducted using *plugged or unplugged* teaching methods. In addition, we investigated whether the students *worked collaboratively* or not because collaboration can have effects on the students' performance [66]. Furthermore, interventions on debugging might have an effect on the debugging process. To this end, we also assessed whether an *intervention was included* in the study.

RQ2: What data sources have been used to capture debugging processes? Second, as the data type is an important foundation for investigating processes [3], we analyzed the data that was used for capturing debugging processes.

RQ3: How were debugging processes analyzed? Third, we investigated how the data was analyzed (qualitative vs. quantitative) and examined the methodologies in detail. We describe the analysis methodologies because they provide deep insights into previous approaches, which offer inspiration for further process analysis.

RQ4: Which aspects have been analyzed within the captured debugging processes? Fourth, we characterized the studies according to the aspects that have been taken into account. To this end, we outline the *goal* of the analyses, the *dimensions of the processes* that were analyzed, and the *outcomes and constructs* that were investigated. By analyzing the goal of the studies we aim to provide insights into the purposes of the analyses. Considering the dimensions, we assessed whether the analysis targeted *cognitive*, *meta-cognitive*, *motivational-affective*, or *social* processes [3]. Those dimensions offer the chance to classify which kind of processes have been analyzed within the studies in our corpus. Moreover, investigating outcomes and constructs analyzed in the studies provides deep insights into which topics were taken into account. As a result, this offers the possibility to derive aspects that are still unaddressed in this field of research.

3.1 Paper Selection

In general, we followed the process proposed by the JBI Manual for Evidence Synthesis for conducting scoping reviews [52]. Deviations from this approach are described and justified at the respective step. For reporting, we applied the PRISMA guidelines for Scoping Reviews [59].

First, we defined the following inclusion criteria for selecting papers that addressed our goal and the RQs best:

- **Age Group:** K-12 or higher education students
- **Topic:** Work that focuses on debugging processes

Considering *debugging processes*, we used a broad understanding and included all papers that addressed debugging processes independently of the analysis method and focus. To be included in the corpus of this scoping review, a study had to meet both inclusion criteria. All other studies were excluded.

For the initial paper selection, we derived from the proposed methodology of the JBI Manual because it did not consider already existing bodies of knowledge. Our paper selection process is visualized in Figure 1. We started with **12 base papers** that the authors knew from previous work. In the first round of paper selection in April 2025, we searched Google Scholar using the terms *debugging process education*, *debugging process computing education*, and *debugging computing education*, which we derived from the keywords of the base papers. Questionable papers were examined and discussed in detail by both authors, who consequently agreed on inclusion or exclusion.

As a result, we excluded papers like the work from Vessey (1985) because she focused on analyzing debugging processes of professional software developers, which was not within our targeted age group. Moreover, the paper from Hassan et al. (2024) was excluded because they analyzed the usage of debuggers for enhancing code comprehension without capturing debugging processes. Comparably, the paper from Michaeli and Romeike (2019) was not included in the review, because they only presented a structured debugging process as part of an intervention but did not investigate the process itself. Finally, we concluded the first round of paper selection with **21 papers**.

Subsequently, for the second round of paper selection, we conducted one round of forward and backward snowballing according to the guidelines of Wohlin (2014) in June 2025. For forward snowballing, we used Google Scholar to assess the references to the

included papers. In total, we identified **160 papers** for further examination. Those papers were also assessed by the authors, who again applied the inclusion criteria stated at the beginning of this section. This was followed by intense discussions about questionable papers, resulting in agreement on inclusion or exclusion. After the second round of paper selection, we added further **15 papers** to the corpus. As a result, the corpus of this scoping review consists of **36 papers**. Table 3 provides an overview of all the papers that were included in the scoping review.

3.2 Data Extraction

To analyze the corpus, we defined categories aligned with the four research questions. For RQ1 and RQ4, categories were first derived from a small random sample of the papers to ensure that they reflected the data. For RQ2 and RQ3, there was no need for a subdivision of the categories to answer the research questions. We derived the codes for all but one category inductively from the corpus to ensure our analysis covered all aspects of the included papers. Only the codes for describing the dimensions of the process analysis (RQ4) were taken from existing work: Based on the framework from Bauer et al. (2025), we distinguished between cognitive, meta-cognitive, social, and motivational-affective processes. Table 1 provides an overview of all research questions, their categories, and the corresponding codes.

Both authors carried out data extraction collaboratively. All coding decisions and interpretations were discussed and agreed on. In Table 2 we present an excerpt of the analysis table of the included studies.

Bibliometric Information: Year of Publication, Age Group	
RQ1: How was the study designed to capture debugging processes?	
Code for Debugging	Own, Others, No code
Code Representation	Plugged, Unplugged, No code
Collaboration	Yes, No
Intervention Included	Yes, No
RQ2: What data sources have been used to capture debugging processes?	
Data Sources	Screen recordings, Videos, Snapshots, Retrospective interviews, Think-aloud, Artifacts, Survey, Performance test, Eyetracking, Interviews, Log data, Manual logs
RQ3: How were debugging processes analyzed?	
Analysis Method	Qualitative-inductive, Qualitative-deductive, Quantitative, Qualitative & Quantitative
RQ4: Which aspects have been analyzed within the captured debugging processes?	
Goal	Describe, Compare, Intervene
Dimensions	Cognitive, Meta-cognitive, Social, Motivational-affective
Aspects	Debugging strategies, Debugging performance, Debugging process-steps, Bugs, Emotions, Collaboration

Table 1: Categories and codes used for answering the research questions of the scoping review.

4 Results

The following section presents the results of this scoping review. After describing some bibliometric information about the corpus, we answer the research questions.

Our corpus consists of **36 papers**, which investigated debugging processes using various data types and focused on several aspects. The oldest paper in the corpus was published in 1986 [61], while the majority of the papers were published between 2020 and 2025 (see Figure 2). As we included only papers targeting K-12 and higher education students, we could identify the following four different age groups: pre-school, primary school, secondary school, and higher education (see Figure 3). Thereby, pre-school includes all children before first grade, primary school ranges from first to fourth grade, and secondary school includes grades five and up.

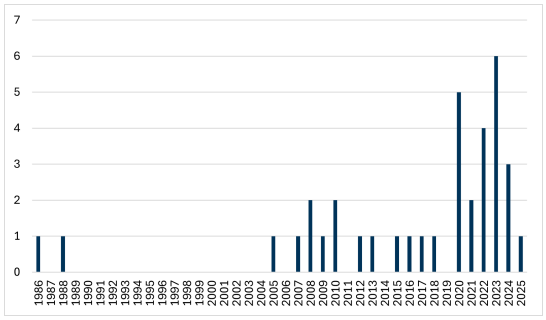

Figure 2: Years of Publication of the included papers.

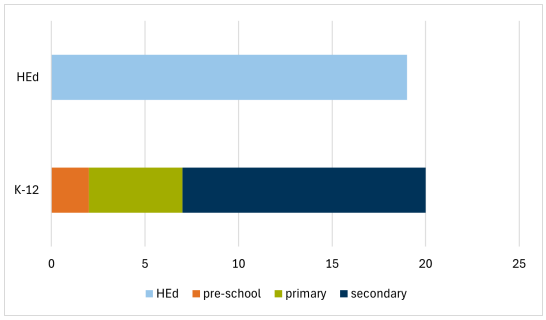

Figure 3: Age group of the study participants.

4.1 RQ1: Study Design

For analyzing the study designs used for investigating debugging processes, we evaluated the following four aspects: selection of the code, plugged vs. unplugged task design, collaboration vs. individual work, and whether an intervention was included or not.

4.1.1 Selection of Code. One important aspect to consider when designing studies to analyze debugging processes is the selection of the code that the students should debug. Usually, in the classroom, when learning to program, students debug their own code, while debugging tasks frequently ask students to debug code written by someone else. Furthermore, others' code causes additional

Source	Year of publication	Age group	Data source	Analysis methodology	Plugged vs. unplugged	Prior intervention	Collaborative work	Code	Goal	Dimensions	Aspects
[48]	2024	K–12	Screen recording Videos	Qualitative, deductive	Plugged Unplugged	No	Yes	Own code	describe compare	Cognitive Social	Collaboration Process Steps
[62]	2023	HEd	Screen recording Think-Aloud	Qualitative, inductive	Plugged	Yes	No	Others' code	describe	Cognitive	Debugging Strategies
...											

Table 2: Excerpt of the analysis table used for investigating the papers included to the corpus.

hurdles in the debugging process, as the students first need to understand the program before they can start debugging. 17 studies in our corpus (e.g., [4, 24, 65]) captured the students when working on programming exercises and then focused on the debugging sequences for analysis. This allows for recording the most natural debugging process as students are debugging their own code. In contrast, 17 other papers (e.g., [14, 22, 62]) used specific debugging tasks which were designed by others. Knowing common bugs in advance not only supports teachers but also allows for better comparisons between students' debugging processes. The work from Kocabas et al. (2022) presents a unique case, working with Lego bricks instead of a traditional representation of code [32]. However, we interpreted the given Lego building as an unplugged type of code and classified it as working with others' code. Moreover, two papers in our corpus did not use code for their study at all [15, 41]. They collected their data via interviews with the students, discussing how they would approach a debugging task (see Figure 4).

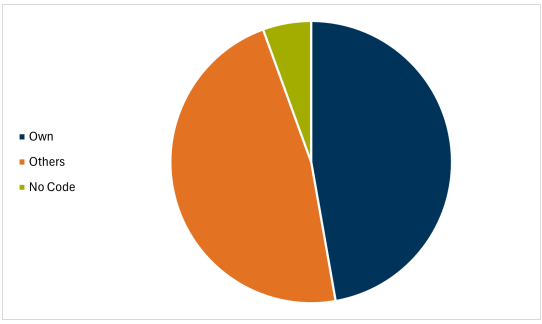

Figure 4: Type of code that was used for the study: own code, others' code, no code.

4.1.2 Plugged vs. Unplugged Task Design. Another important aspect to consider is how the task is designed, whether it is a plugged or an unplugged task. In general, the majority of the papers used common plugged activities. On the one hand, they addressed typical text-based programming languages (e.g., [5, 35, 56]), and on the other hand, block-based programming languages were addressed as well (e.g., [4, 30, 65]). Two contributions also considered debugging e-textiles, which are a special case of physical computing devices [22, 23]. However, especially in pre-school and primary school, working unplugged is an important teaching approach in computing education, which, for example facilitates better program comprehension. In our corpus, four studies used unplugged activities. Three of them had the typical age group of pre-school or primary school students in their scope [32, 45, 48]. For example, Kocabas et al. (2022) investigated debugging strategies that young children used for fixing Lego buildings so that they matched the required structure [32]. Only one study worked unplugged with university students, using paper-based program code for theoretically describing debugging processes [63]. Moreover, one paper described a study using plugged and unplugged tasks, as they were working with primary and secondary school students [48]. Figure 5 shows the crossover of the age group with the task design.

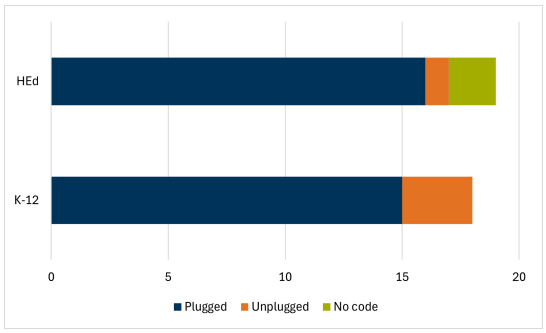

Figure 5: Plugged vs. Unplugged task design in relation with the age group of the respective study.

4.1.3 Collaboration vs. Individual Work. Moreover, distinguishing between collaborative and individual work is another relevant aspect for designing tasks. Collaboration offers the students the possibility to work together [57] and to share cognitive load [7]. Especially, in programming contexts, collaborative settings have been intensely investigated [19], while in the context of debugging is a lack of interventions that target collaboration [66]. About a third of the studies in our corpus asked the students to work collaboratively in pairs (e.g., [23, 46, 48]) or in small groups (e.g., [13, 29]), while only some of them later also assessed effects of collaboration [13, 22, 23, 46, 48, 61] (see Sections 4.4.2 and 4.4.3). In contrast, the 24 remaining studies required individual work (see Figure 6).

4.1.4 Intervention Included? As interventions can affect the outcome of a study, the fourth aspect we investigated regarding study design was whether the study had an intervention included, or not. In our corpus, we identified 14 studies (e.g., [11, 24, 62]) that described an intervention (see Figure 7). Interventions, for instance,

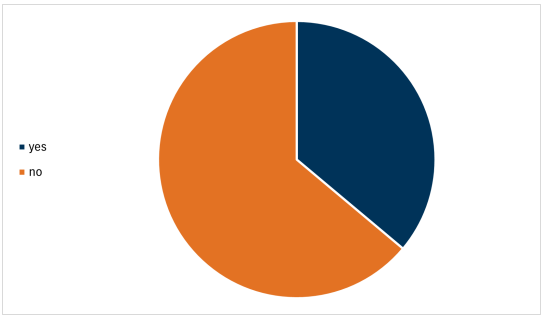

Figure 6: Proportion of included papers in which the students worked collaboratively.

used the analysis of worked examples [4] or scaffolding [29] to improve students' debugging skills. However, not all of them later assessed the outcome of the intervention (see Section 4.4.1). Figure 7 shows the overlap between studies that included an intervention and those that also assessed the intervention's outcome.

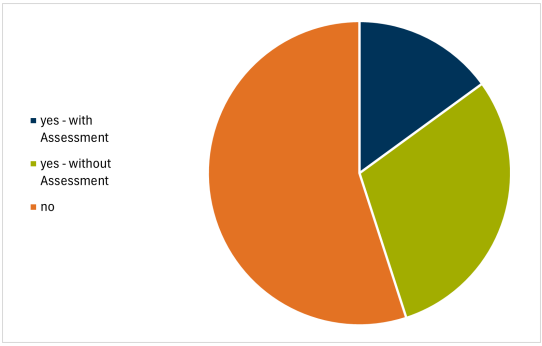

Figure 7: Proportions of papers that included an intervention and also assessed the outcomes.

4.2 RQ2: Data Sources

The data source is one of the most crucial aspects for analyzing processes, as the data is the foundation for the subsequent analysis and the outcomes of the study. We identified 12 different types of data that were used for capturing debugging processes:

- *Screen recordings:* Videos that only capture the students screens and conversations
- *Videos:* Recordings that show the whole classroom and/or students working on tasks
- *Snapshots:* Copy of the code that is captured at distinct points of the process, e.g., every time the code was uploaded to an auto-grader
- *Retrospective-Interviews:* An interview that is conducted after working on the debugging task to gain more information about the process
- *Think-Aloud:* Audio recordings of the students verbalizing their thoughts while debugging
- *Artifacts:* Final product (code, debugging diary,...) of the students after debugging
- *Survey:* Questionnaire on aspects related to debugging
- *Performance Test:* Test that assesses the debugging performance of students
- *Eyetracking:* Data that captures the students eye movements during debugging
- *Interview:* An interview that is conducted to collect information on the students' debugging approaches
- *Log Data:* Data that is collected automatically by a system and offers the possibility to reconstruct the whole debugging process with all interim steps
- *Manual Logs:* Notes from the researchers observing students when debugging

In the following, we dive deeper into selected types of data that might not be easy to distinguish from others: *Screen recordings* were used the most in our corpus to record the students' debugging processes (e.g., [8, 10, 31]). In contrast to screen recordings, *videos* captured the whole classroom or the students when working on the tasks (e.g., [22, 29, 46]). With 10 studies, videos were the second most used data source within our corpus.

Distinguishing between interviews and retrospective interviews, a study was coded as an *interview* when the interview was the manner in which the debugging process was captured [30, 41]. When the students were interviewed after working on the debugging tasks, in order to gain more information about their approach, how they did, what problems they encountered, etc., we coded the interviews as *retrospective interviews* (e.g., [15, 22, 28]).

Moreover, we differentiated between *program snapshots* and *log data*. Both data types are typically collected from IDEs or auto-graders. However, log data captures all actions that occur within the IDE, offering a gapless documentation of the debugging process. In contrast, program snapshots feature the state of debugging only at specific points, like compilation or uploading to an auto-grader. Consequently, snapshots lack all actions that are conducted between two points in time. Within our corpus, only two papers captured log data [35, 56], while seven studies worked with program snapshots (e.g., [1, 39, 40]).

Considering the number of data sources per study, in several studies, more than one type of data was collected and analyzed. For example, screen recordings were often combined with retrospective interviews or think-aloud protocols (e.g., [8, 11, 28]). Moreover, in some papers, screen recordings were augmented by videos that captured the classroom situation (e.g, [30, 33, 46]). In contrast, snapshots were used all the time on their own. More details about the frequency of the data sources and the number of studies that used more than one data source are provided in Figures 8 and 9.

4.3 RQ3: Data Analysis

To gain insights into how the data was analyzed, we investigated the methodologies applied in the papers. An analysis was either qualitative, quantitative, or used both approaches.

Within our corpus, we found 32 papers that conducted a **qualitative** analysis. In total, 20 studies applied an *inductive* category formation, analyzing several different outcomes and constructs (refer to Section 4.4.3). Those papers mainly investigated debugging strategies (e.g., [8, 16, 25]). A few also took debugging steps into account (e.g., [24, 39, 63]). 17 studies used a *deductive* approach

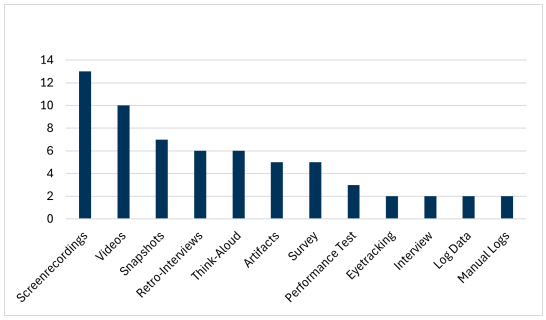

Figure 8: Absolute numbers of data that was used to capture the debugging process. Several papers used more than one data source.

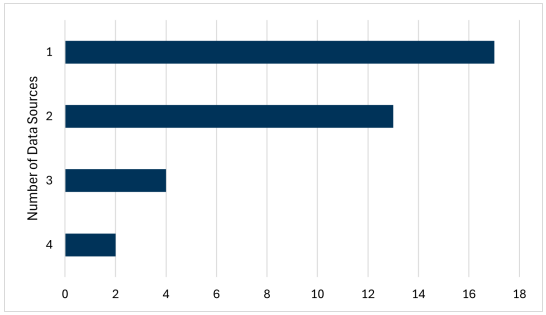

Figure 9: Number of Data Sources used in a paper.

referring to existing literature for developing a category system e.g., [4, 13, 48]). The theory that was used was, in most cases, a list of established debugging strategies (e.g., [25, 47, 60]). Other theories used for deductive analysis were related to collaboration [22], regulation strategies [11, 13, 48], and some studies mapped their data to debugging process steps [10, 30, 31, 48]. Moreover, five papers within our corpus used both inductive and deductive approaches to answer the research questions (e.g., [11, 30, 61]).

Focusing on **quantitative** data analysis, four papers fully relied on quantitative methods [37, 38, 56, 68] while 13 papers combined qualitative and quantitative approaches (e.g., [4, 40]). The majority of them investigated surveys or program snapshots (e.g., [16, 39, 40]). However, some of the studies analyzed eyetracking data quantitatively [35, 37]. For an overview of the proportions of the methodological approaches used, refer to Figure 10. As some papers used more than one approach, the total number of papers in the figure exceeds 36.

In general, in our corpus, there is a lack of consistent approaches for analyzing the data. We did not find the same methodology applied twice by different research groups. Each research group described its own methodology. The following two examples highlight the variety of methods used to analyze debugging processes and provide insights into how such analyses are conducted: Parkinson et al. (2024) started with transcribing all audio, video, and screen recording data verbally. For coding the data, they applied a qualitative-deductive approach using debugging process steps and types of regulation strategies from the literature. As a last step,

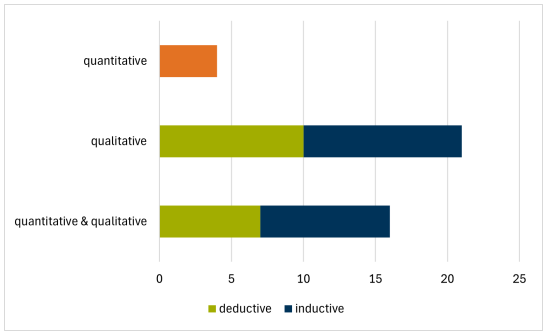

Figure 10: Type of methodology: Qualitative vs. quantitative.

they attempted to identify patterns in the codings [48]. In contrast, Liu and Paquette (2023) conducted a quantitative analysis by calculating "mean, median, and standard deviation of each debugging feature" using log data in order to gain information about the students' debugging behavior. Moreover, by using logistic regression models, they classified the students' approaches as either efficient or inefficient [38].

4.4 RQ4: Analysis Aspects

To investigate *what* aspects the papers in our corpus analyzed, we captured the overall goal of the analysis together which the analyzed dimensions, and outcomes and constructs.

4.4.1 Goal. To classify the goal, we grouped our corpus into three categories: **describe**, **compare**, and **intervene**. Even if the categories were derived from the corpus, they are not disjunct, and a study could address more than one goal. Overall, most studies focused solely on describing the analysis results, which often were debugging strategies (e.g., [5, 47, 62]) or the debugging performance of the students (e.g., [23, 28, 35]). However, debugging performance was also often analyzed by comparing metrics such as time spent and debugging success [1, 14] or by contrasting the eye movements of high and low performing students [37]. Only four papers in our corpus aimed at assessing the effect of interventions [4, 29, 31, 61]. Besides investigating the intervention success, the work from Misirli and Komis (2023) also aimed at describing other outcomes like strategies or typical bugs [45]. Furthermore, the paper from Zhang et al. (2023) analyzed the debugging performance of students by describing their performance in detail, contrasting different groups of students, and investigating the effects of an intervention on the performance [68]. For the exact distribution, consider Figure 11.

4.4.2 Dimensions. For characterizing processes, we considered four dimensions:

- *cognitive processes:* mental processes that are needed for learning and problem-solving [2]
- *meta-cognitive processes:* processes that supervise the cognitive processes using strategies [36]
- *motivational-affective processes:* emotional and motivational states during debugging [9]
- *social processes:* collaborative processes during debugging [12]

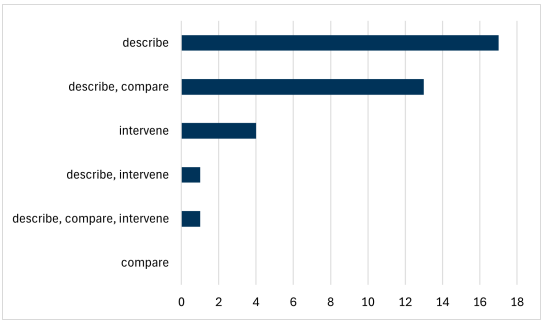

Figure 11: Goal of the analyses.

In our corpus, we could identify all four dimensions, and each study addressed at least one dimension. Eight studies also address more than one dimension (e.g., [39, 48, 61]). However, the vast majority focuses on analyzing cognitive processes like the usage of strategies (e.g., [32, 41, 65]), typical bugs that occur in students' code (e.g., [14, 30, 45]), or the students' debugging performance (e.g., [29, 35, 38]). At least some papers took social processes like the effect of group dynamics [13, 22, 23, 46, 61] or the application of regulation strategies in collaborative settings [48] into account. Considering motivational-affective processes, five papers evaluated emotions in the debugging process [8, 11, 16, 22] and self-perceptions on debugging [39]. Only two papers in our corpus investigated meta-cognition by investigating planning [61] and regulation processes when debugging [13]. Figure 12 provides details about the absolute numbers of papers per dimension.

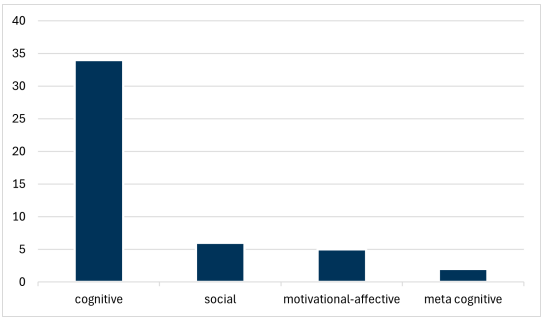

Figure 12: Dimensions that were investigated with the analyses.

4.4.3 Outcomes & Constructs. For investigating outcomes and constructs that were analyzed by the papers in our corpus, we inductively identified six different codes:

- *Debugging Strategies:* Debugging Strategies that the students used while identifying and fixing bugs
- *Debugging Performance:* Measurement for how good, fast, etc. the students performed in debugging
- *Debugging Process-Steps:* Steps from ideal-typical debugging processes that were investigated in the students' processes
- *Bugs:* Bugs that occurred in students' programs
- *Emotions:* Emotions that arise during the debugging process
- *Collaboration:* Impact of collaboration on the debugging process

The majority of our corpus focused on investigating debugging strategies in the students' work. The strategies that the students use are mostly related to steps in the process of identifying and fixing bugs. Popular examples are code tracing for a better understanding of the program [14, 15, 41, 47, 62], pattern matching to e.g., detect missing curly braces [14, 15, 41, 47], and using print statements for examining variable values [62]. Furthermore, debugging strategies were frequently classified as effective or ineffective (e.g., [14, 47, 62]). The second-most analyzed outcome of the studies within our corpus was the debugging performance of the students. Some studies investigated the relationship between programming and debugging skills [1, 14], while others focused on differences in the debugging behavior of high and low performing students [37, 39]. Only a few studies took the the remaining aspects into account: Regarding debugging process-steps, the studies often analyzed the debugging behavior of the students (e.g., [33, 56, 63]). Only three papers mapped the students' work to established debugging process steps [10, 30, 48]. Considering bugs, some papers investigated which bugs frequently arise in the students' code, e.g., incorrect variable definitions, or wrong conditional statements [1, 4, 30, 45] and how difficult they were to solve [14]. Regarding collaboration, only a few studies assessed the effects of group work on debugging processes [13, 22, 23, 46, 61] and the application of social regulation strategies [48]. Within our corpus, five papers considered emotions like anxiety [16], joy [16], frustration [11, 16], and the attitudes towards debugging [8, 22, 39]. The distribution is visualized in Figure 13.

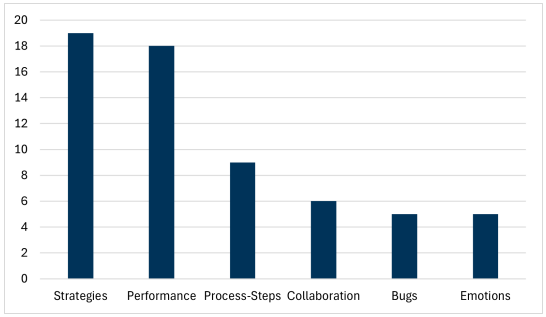

Figure 13: Aspects on which the analyses focused. An analysis could focus on one or more aspects.

5 Discussion

In this scoping review, we investigated 36 papers that analyzed the debugging processes of K-12 and higher education students. We analyzed how the studies were designed, which data were captured, what analysis methods were used, and what aspects were investigated in order to synthesize previous research on debugging processes. In the following, we present and discuss themes (and non-themes) that emerged from our corpus.

The papers in our corpus **primarily analyzed cognitive dimensions** of the debugging processes, thereby focusing on investigating debugging strategies and assessing the debugging performance of students. Only two papers [13, 61] focused on *meta-cognitive strategies* like planning, monitoring, and reflecting on the debugging processes. This result is surprising to us, because meta-cognitive skills are a key factor for successful debugging [66]. Nevertheless, this goes along with the conclusion of Yang et al. (2024) that there is a need for more interventions focusing on non-cognitive constructs.

Besides a lack of meta-cognitive aspects, *motivational-affective* and *social aspects* also received limited attention. Only a few papers in our corpus analyzed aspects like attitudes towards debugging [8] or which emotions students experience during debugging [16]. However, as emotions are important, especially regarding the level of motivation and frustration, this lack is emerging. Taking emotions into account offers the possibility to develop more holistic models of students' debugging behavior, resulting in better emotional support for students and reducing frustration. Focusing on social aspects, especially in the context of programming education, the approach of pair programming is widely explored [19]. In contrast, despite the great potential that collaborative processes offer for reducing cognitive load and improved problem solving processes, collaborative debugging remains underexplored. Even if aspects like collaborative regulation strategies and effects of group dynamics on debugging processes were considered in a few studies, collaboration is only a side aspect when investigating debugging processes.

In our corpus, a vast variety of **data sources** is used. However, not all data types are suitable for a process analysis as they do not capture the whole process. One popular example of this are program snapshots that were mostly collected via auto-grading platforms and were used in several papers of our corpus. Those snapshots omit what happens between two submissions, and consequently they miss several relevant debugging steps, like formulating and verifying a hypothesis. Consequently, it is not possible to investigate ideal-typical debugging processes using snapshots. In contrast, data collected via programming environments (log data) and screen recordings offer the possibility to capture complete processes. While screen recordings were frequently used already, studies using log data to investigate debugging behavior are rare. Even when log data or screen recordings were captured, the analysis often focused on selected time frames or on prominent events, without investigating the whole debugging process. As a result, often no real process analysis was conducted because either the data did not facilitate this or the research focus was on other aspects than process analysis. Moreover, investigating the whole debugging process by mapping log data or screen recordings to debugging process-steps and contrasting them with an ideal-typical debugging process, offers the chance to enhance our understanding of how students debug and when they need support. In addition, log data offers the foundation for personalized and automated real-time feedback for learners. Consequently, it is a missed opportunity not to use log data to analyze debugging processes. Going along with a lack of process analysis, investigating debugging is often reduced to analyzing performance outcomes, e.g., the number of bugs fixed or the time spent on a task. Especially for intervention studies, assessing debugging processes offers great potential for evaluating the students' learning progress instead of only analyzing the learning outcome. However, we could not identify any studies that measured the success of an intervention by analyzing processes.

For **data analysis**, the majority of the studies in our corpus used an inductive approach. This result might be grounded in the history of debugging: Debugging was developed as a necessity for software development, and the theory was later inductively derived from existing processes. While this is helpful for exploring a new field of research, pure inductive approaches limit the explanatory power of the findings. In contrast, several papers grounded their analysis in prior research, which offers great potential for comparison and deeper insights into debugging processes. However, most of the deductive analyses did not consider the debugging process itself. They rather focused on several aspects related to debugging, like regulation strategies and debugging strategies. Interestingly, even if investigating debugging strategies has so far been a focus of research on debugging, there exists no comprehensive, deductively validated overview of debugging strategies that students apply.

One of the most prominent findings of our review is the **lack of standardized approaches** for analyzing debugging processes. Numerous studies have addressed debugging behavior, but each paper applied its own methodology tailored to the specific study design and data set. We did not identify a single methodology that was applied across more than one research group. Although this variety shows the creativity of the research field, it limits the comparability of the results across studies.

6 Conclusion

This paper presented a scoping review on how debugging processes have been investigated in educational contexts. We identified that typical analysis approaches use a qualitative methodology that describes cognitive aspects of debugging processes, like the application of debugging strategies or the debugging performance of students. In addition, we developed a reporting scheme that can be applied to further debugging process analyses.

Moreover, this scoping review identified several research gaps, such as a lack of log data analysis, which would facilitate the investigation of the whole debugging process, in contrast to snapshot-based approaches. To this end, log data analyses offer a starting point for automated and personalized feedback to the students while debugging.

Another possible future research direction is to assess improved debugging not only as a performance-oriented goal but also as a process-oriented outcome. By doing so, intervention studies should also take changes in the processes as an indicator for learning improvement into account.

Acknowledgments

During the preparation of this work, the authors used ChatGPT 4 and deepLWrite in order to revise the draft for readability and language. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Author(s)	Title	Year	Citation
Ahmadzadeh et al.	"An analysis of patterns of debugging among novice computer science students"	2005	[1]
Bofferding et al.	"The effect of play and worked examples on first and third graders' creating and debugging of programming algorithms"	2022	[4]
Bottcher et al.	"Debugging students' debugging process"	2016	[5]
Chen et al.	"Novices' Debugging Behaviors in VB Programming"	2013	[8]
DeLiema et al.	"Debugging Pathways: Open-Ended Discrepancy Noticing, Causal Reasoning, and Intervening"	2024	[10]
DeLiema et al.	"A Multi-dimensional Framework for Documenting Students' Heterogeneous Experiences with Programming Bugs"	2023	[11]
Emara et al.	"Examining students' debugging and regulation processes during collaborative computational modeling in science"	2020	[13]
Fitzgerald et al.	"Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers"	2008	[14]
Fitzgerald et al.	"Debugging From the Student Perspective"	2010	[15]
Gale and Sentance	"Investigating the Attitudes and Emotions of K-12 Students Towards Debugging"	2023	[16]
Jayathirtha et al.	"Distributed debugging with electronic textiles: understanding high school student pairs' problem-solving strategies, practices, and perspectives on repairing physical computing projects"	2024	[22]
Jayathirtha et al.	"Pair debugging of electronic textiles projects: Analyzing think-aloud protocols for high school students' strategies and practices while problem solving"	2020	[23]
Jemmali et al.	"MAADS: Mixed-Methods Approach for the Analysis of Debugging Sequences of Beginner Programmers"	2020	[24]
Katz and Anderson	"Debugging: an analysis of bug-location strategies"	1987	[25]
Kim et al.	"Revisiting Analogical Reasoning in Computing Education: Use of Similarities Between Robot Programming Tasks in Debugging"	2023	[28]
Kim et al.	"Debugging behaviors of early childhood teacher candidates with or without scaffolding"	2022	[29]
Kim et al.	"Debugging during block-based programming"	2018	[30]
Klahr and Carver	"Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer"	1988	[31]
Kocabas et al.	"The role of color in debugging LEGO structures"	2022	[32]
Lewis	"The importance of students' attention to program state"	2012	[33]
Li et al.	"Task-oriented Analysis on Debugging Process Based on Eye Movements and IDE Interactions"	2021	[35]
Lin et al.	"Tracking Students' Cognitive Processes During Program Debugging—An Eye-Movement Approach"	2016	[37]
Liu and Paquette	"Using submission log data to investigate novice programmers' employment of debugging strategies"	2023	[38]
Liu et al.	"Understanding problem solving behavior of 6–8 graders in a debugging game"	2017	[39]
Mansur et al.	"Exploring the Bug Investigation Techniques of Intermediate Student Programmers"	2020	[40]
McCartney et al.	"Successful students' strategies for getting unstuck"	2007	[41]
Misirli and Komis	"Computational thinking in early childhood education: The impact of programming a tangible robot on developing debugging knowledge"	2023	[45]
Murphy et al.	"Pair debugging: a transactive discourse analysis"	2010	[46]
Murphy et al.	"Debugging: the good, the bad, and the quirky – a qualitative analysis of novices' strategies"	2008	[47]
Parkinson et al.	"Exploring debugging processes and regulation strategies during collaborative coding tasks among elementary and secondary students"	2024	[48]
Shynkarenko and Zhevaho	"Development of a toolkit for analyzing software debugging processes using the constructive approach"	2020	[56]
Webb et al.	"Problem-Solving Strategies and Group Processes in Small Groups Learning Computer Programming"	1986	[61]
Whalley et al.	"A Think-Aloud Study of Novice Debugging"	2023	[62]
Whalley et al.	"Analysis of a Process for Introductory Debugging"	2021	[63]
Yan et al.	"How Do Elementary Students Apply Debugging Strategies in a Block-Based Programming Environment?"	2025	[65]
Zhang et al.	"Combining latent profile analysis and programming traces to understand novices' differences in debugging"	2023	[68]

Table 3: Overview of all papers included to the corpus.

References

[1] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. An analysis of patterns of debugging among novice computer science students. *ACM SIGCSE Bulletin*, 37, 3, 84–88. doi:10.1145/1151954.1067472.

[2] American Psychological Association. 2025. Cognitive Process. Retrieved July 18, 2025 from https://dictionary.apa.org/cognitive-process.

[3] Elisabeth Bauer et al. 2025. Personalizing simulation-based learning in higher education. *Learning and Individual Differences*, 122, 102746. doi:10.1016/j.lindif.2025.102746.

[4] Laura Bofferding, Sezai Kocabas, Mahtob Aqazade, Ana-Maria Haiduc, and Lizhen Chen. 2022. The effect of play and worked examples on first and third graders' creating and debugging of programming algorithms. In *Computational Thinking in PreK-5*. Anne Ottenbreit-Leftwich and Aman Yadav, (Eds.) ACM, New York, NY, USA, 19–29. ISBN: 9781450396158. doi:10.1145/3507951.3519284.

[5] Axel Bottcher, Veronika Thurner, Kathrin Schlierkamp, and Daniela Zehetmeier. 2016. Debugging students' debugging process. In *2016 IEEE Frontiers in Education Conference (FIE)*. 2016 IEEE Frontiers in Education Conference (FIE) (Erie, PA, USA). IEEE, 1–7. ISBN: 978-1-5090-1790-4. doi:10.1109/FIE.2016.7757447.

[6] Gert Braune and Andreas Mühling. 2024. Observing Students' Behavior During Problem Solving: Determinants of Success. In *Proceedings of the 19th WiPSCE Conference on Primary and Secondary Computing Education Research*. WiPSCE '24: The 19th WiPSCE Conference on Primary and Secondary Computing Education Research (Munich Germany). Tilman Michaeli, Sue Sentance, and Nadine Bergner, (Eds.) ACM, New York, NY, USA, 1–10. ISBN: 9798400710056. doi:10.1145/3677619.3678109.

[7] Sallyann Bryant, Pablo Romero, and Benedict Du Boulay. 2008. Pair programming and the mysterious role of the navigator. *International Journal of Human-Computer Studies*, 66, 7, 519–529. doi:10.1016/j.ijhcs.2007.03.005.

[8] Mei-Wen Chen, Cheng-Chih Wu, and Yu-Tzu Lin. 2013. Novices' Debugging Behaviors in VB Programming. In *2013 Learning and Teaching in Computing and Engineering*. 2013 Learning and Teaching in Computing and Enginering (LaTiCE) (Macau). IEEE, 25–30. ISBN: 978-1-4673-5627-5. doi:10.1109/LaTiCE.2013.38.

[9] Edward L. Deci and Richard M. Ryan. 2004. *Handbook of self-determination research*. University Rochester Press.

[10] David DeLiema, Jeffrey K. Bye, and Vijay Marupudi. 2024. Debugging Pathways: Open-Ended Discrepancy Noticing, Causal Reasoning, and Intervening. *ACM Transactions on Computing Education*, 24, 2, 1–34. doi:10.1145/3650115.

[11] David DeLiema, Yejin Angela Kwon, Andrea Chisholm, Immanuel Williams, Maggie Dahn, Virginia J. Flood, Dor Abrahamson, and Francis F. Steen. 2023. A Multi-dimensional Framework for Documenting Students' Heterogeneous Experiences with Programming Bugs. *Cognition and Instruction*, 41, 2, 158–200. doi:10.1080/07370008.2022.2118279.

[12] Ilana Dubovi and Iris Tabak. 2020. An empirical analysis of knowledge co-construction in YouTube comments. *Computers & Education*, 156, 103939. doi:10.1016/j.compedu.2020.103939.

[13] Mona Emara, Shuchi Grover, Nicole Hutchins, Gautam Biswas, and Caitlin Snyder. 2020. Examining students' debugging and regulation processes during collaborative computational modeling in science.

[14] Sue Fitzgerald, Gary Lewandowski, Renée McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, 18, 2, 93–116. doi:10.1080/08993400802114508.

[15] Sue Fitzgerald, Renée McCauley, Brian Hanks, Laurie Murphy, Beth Simon, and Carol Zander. 2010. Debugging From the Student Perspective. *IEEE Transactions on Education*, 53, 3, 390–396. doi:10.1109/TE.2009.2025266.

[16] Laurie Gale and Sue Sentance. 2023. Investigating the Attitudes and Emotions of K-12 Students Towards Debugging. In *The United Kingdom and Ireland Computing Education Research (UKICER) conference*. UKICER 2023: The United Kingdom and Ireland Computing Education Research conference (Swansea Wales Uk). Troy Astarte, Faron Moller, Keith Quille, and Seán Russell, (Eds.) ACM, New York, NY, USA, 1–7. ISBN: 9798400708763. doi:10.1145/3610969.3611120.

[17] David J. Gilmore. 1991. Models of debugging. *Acta Psychologica*, 78, 1-3, 151–172. doi:10.1016/0001-6918(91)90009-O.

[18] John D. Gould. 1975. Some psychological evidence on how people debug computer programs. *International Journal of Man-Machine Studies*, 7, 2, 151–182. doi:10.1016/S0020-7373(75)80005-8.

[19] Brian Hanks, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. 2011. Pair programming in education: a literature review. *Computer Science Education*, 21, 2, 135–173. doi:10.1080/08993408.2011.579808.

[20] Mohammed Hassan, Grace Zeng, and Craig Zilles. 2024. Evaluating How Novices Utilize Debuggers and Code Execution to Understand Code. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*. ICER 2024: ACM Conference on International Computing Education Research (Melbourne VIC Australia). Paul Denny, Leo Porter, Margaret Hamilton, and Briana Morrison, (Eds.) ACM, New York, NY, USA, 65–83. ISBN: 9798400704758. doi:10.1145/3632620.3671126.

[21] Nicole Heitzmann, Matthias Stadler, Constanze Richters, Anika Radkowitsch, Ralf Schmidmaier, Marc Weidenbusch, and Martin R. Fischer. 2023. Learners' adjustment strategies following impasses in simulations - Effects of prior knowledge. *Learning and Instruction*, 83, 101632. doi:10.1016/j.learninstruc.2022.101632.

[22] Gayithri Jayathirtha, Deborah Fields, and Yasmin Kafai. 2024. Distributed debugging with electronic textiles: understanding high school student pairs' problem-solving strategies, practices, and perspectives on repairing physical computing projects. *Computer Science Education*, 34, 4, 718–752. doi:10.1080/08993408.2023.2297738.

[23] Gayithri Jayathirtha, Deborah Fields, and Yasmin Kafai. 2020. Pair debugging of electronic textiles projects: Analyzing think-aloud protocols for high school students' strategies and practices while problem solving.

[24] Chaima Jemmali, Erica Kleinman, Sara Bunian, Mia Victoria Almeda, Elizabeth Rowe, and Magy Seif El-Nasr. 2020. MAADS: Mixed-Methods Approach for the Analysis of Debugging Sequences of Beginner Programmers. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE '20: The 51st ACM Technical Symposium on Computer Science Education (Portland OR USA). Jian Zhang, Mark Sherriff, Sarah Heckman, Pamela Cutter, and Alvaro Monge, (Eds.) ACM, New York, NY, USA, 86–92. ISBN: 9781450367936. doi:10.1145/3328778.3366824.

[25] Irvin R. Katz and John R. Anderson. 1987. Debugging: an analysis of bug-location strategies. *Human-Computer Interaction*, 3, 4, 351–399. doi:10.1207/s15327051hci0304_2.

[26] Max Kesselbacher and Andreas Bollin. 2019. Discriminating Programming Strategies in Scratch. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*. WiPSCE'19: 14th Workshop in Primary and Secondary Computing Education (Glasgow Scotland Uk). ACM, New York, NY, USA, 1–10. ISBN: 9781450377041. doi:10.1145/3361721.3361727.

[27] Claudius M. Kessler and John R. Anderson. 1986. A model of novice debugging in LISP. In *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*. Ablex Publishing Corp, USA, 198–212. ISBN: 089391388X.

[28] ChanMin Kim, Emre Dinç, Eunseo Lee, Afaf Baabdullah, Anna Y. Zhang, and Brian R. Belland. 2023. Revisiting Analogical Reasoning in Computing Education: Use of Similarities Between Robot Programming Tasks in Debugging. *Journal of Educational Computing Research*, 61, 5, 1036–1063. doi:10.1177/07356331221142912.

[29] ChanMin Kim, Lucas Vasconcelos, Brian R. Belland, Duygu Umutlu, and Cory Gleasman. 2022. Debugging behaviors of early childhood teacher candidates with or without scaffolding. *International Journal of Educational Technology in Higher Education*, 19, 1. doi:10.1186/s41239-022-00319-9.

[30] ChanMin Kim, Jiangmei Yuan, Lucas Vasconcelos, Minyoung Shin, and Roger B. Hill. 2018. Debugging during block-based programming. *Instructional Science*, 46, 5, 767–787. doi:10.1007/s11251-018-9453-5.

[31] David Klahr and Sharon McCoy Carver. 1988. Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology*, 20, 3, 362–404. doi:10.1016/0010-0285(88)90004-7.

[32] Sezai Kocabas, Laura Bofferding, Yi Zhu, and Yiheng Liang. 2022. The role of color in debugging LEGO structures. In *Proceedings of the 16th International Conference of the Learning Sciences-ICLS 2022, pp. 1449-1452*. International Society of the Learning Sciences.

[33] Colleen M. Lewis. 2012. The importance of students' attention to program state. In *Proceedings of the ninth annual international conference on International computing education research*. ICER '12: International Computing Education Research Conference (Auckland New Zealand). Alison Clear, Kate Sanders, and Beth Simon, (Eds.) ACM, New York, NY, USA, 127–134. ISBN: 9781450316040. doi:10.1145/2361276.2361301.

[34] 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*. ACE'19: Twenty-First Australasian Computing Education Conference (Sydney NSW Australia). Simon and Andrew Luxton-Reilly, (Eds.) ACM, New York, NY, USA, 79–86. ISBN: 9781450366229. doi:10.1145/3286960.3286970.

[35] Xinyu Li, Wei Liu, Huitong Liu, Jing Xu, and Wenqing Cheng. 2021. Task-oriented Analysis on Debugging Process Based on Eye Movements and IDE Interactions. In *2021 16th International Conference on Computer Science & Education (ICCSE)*. 2021 16th International Conference on Computer Science & Education (ICCSE) (Lancaster, United Kingdom). IEEE, 379–384. ISBN: 978-1-6654-1468-5. doi:10.1109/ICCSE51940.2021.9569438.

[36] Lyn Lim, Maria Bannert, Joep van der Graaf, Inge Molenaar, Yizhou Fan, Jonathan Kilgour, Johanna Moore, and Dragan Gašević. 2021. Temporal Assessment of Self-Regulated Learning by Mining Students' Think-Aloud Protocols. eng. *Frontiers in psychology*, 12, 749749. eprint: 34803832. doi:10.3389/fpsyg.2021.749749.

[37] Yu-Tzu Lin, Cheng-Chih Wu, Ting-Yun Hou, Yu-Chih Lin, Fang-Ying Yang, and Chia-Hu Chang. 2016. Tracking Students' Cognitive Processes During Program Debugging—An Eye-Movement Approach. *IEEE Transactions on Education*, 59, 3, 175–186. doi:10.1109/TE.2015.2487341.

[38] Qianhui Liu and Luc Paquette. 2023. Using submission log data to investigate novice programmers' employment of debugging strategies. In *LAK23: 13th International Learning Analytics and Knowledge Conference*. LAK 2023: 13th International Learning Analytics and Knowledge Conference (Arlington TX USA). Isabel Hilliger, Hassan Khosravi, Bart Rienties, and Shane Dawson, (Eds.) ACM, New York, NY, USA, 637–643. ISBN: 9781450398657. doi:10.1145/3576050.3576094.

[39] Zhongxiu Liu, Rui Zhi, Andrew Hicks, and Tiffany Barnes. 2017. Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education*, 27, 1, 1–29. doi:10.1080/08993408.2017.1308651.

[40] Rifat Sabbir Mansur, Ayaan M. Kazerouni, Stephen H. Edwards, and Clifford A. Shaffer. 2020. Exploring the Bug Investigation Techniques of Intermediate Student Programmers. In *Proceedings of the 20th Koli Calling International Conference on Computing Education Research* (Koli Calling '20). Association for Computing Machinery, New York, NY, USA. ISBN: 9781450389211. doi:10.1145/3428029.3428040.

[41] Robert McCartney, Anna Eckerdal, Jan Erik Mostrom, Kate Sanders, and Carol Zander. 2007. Successful students' strategies for getting unstuck. *ACM SIGCSE Bulletin*, 39, 3, 156–160. doi:10.1145/1269900.1268831.

[42] Renée McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: a review of the literature from an educational perspective. *Computer Science Education*, 18, 2, 67–92. doi:10.1080/08993400802114581.

[43] 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)*. 2019 IEEE Global Engineering Education Conference (EDUCON) (Dubai, United Arab Emirates). IEEE, 1030–1038. ISBN: 978-1-5386-9506-7. doi:10.1109/EDUCON.2019.8725282.

[44] Tilman Michaeli and Ralf Romeike. 2019. Improving Debugging Skills in the Classroom - The Effects of Teaching a Systematic Debugging Process. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*. WiPSCE'19: 14th Workshop in Primary and Secondary Computing Education

(Glasgow Scotland Uk). ACM, New York, NY, USA, 1–7. ISBN: 9781450377041. doi:10.1145/3361721.3361724.

[45] Anastasia Misirli and Vassilis Komis. 2023. Computational thinking in early childhood education: The impact of programming a tangible robot on developing debugging knowledge. *Early Childhood Research Quarterly*, 65, 139–158. doi:10.1016/j.ecresq.2023.05.014.

[46] Laurie Murphy, Sue Fitzgerald, Brian Hanks, and Renée McCauley. 2010. Pair debugging: a transactive discourse analysis. In *Proceedings of the Sixth international workshop on Computing education research*. ICER '10: International Computing Education Research Workshop (Aarhus Denmark). Michael E. Caspersen, Mike Clancy, and Kathryn Sanders, (Eds.) ACM, New York, NY, USA, 51–58. ISBN: 9781450302579. doi:10.1145/1839594.1839604.

[47] 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. *ACM SIGCSE Bulletin*, 40, 1, 163–167. doi:10.1145/1352322.1352191.

[48] Meghan M. Parkinson, Seppe Hermans, David Gijbels, and Daniel L. Dinsmore. 2024. Exploring debugging processes and regulation strategies during collaborative coding tasks among elementary and secondary students. *Computer Science Education*, 34, 4, 617–644. doi:10.1080/08993408.2024.2305026.

[49] Reinhard Pekrun and Lisa Linnenbrink-Garcia. 2012. Academic Emotions and Student Engagement. In *Handbook of Research on Student Engagement*. Sandra L. Christenson, Amy L. Reschly, and Cathy Wylie, (Eds.) Springer US, Boston, MA, 259–282. ISBN: 978-1-4614-2017-0. doi:10.1007/978-1-4614-2018-7_12.

[50] D. 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, 37–55. doi:10.2190/GUJT-JCBJ-Q6QU-Q9PL.

[51] 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, 83–110. doi:10.1007/s11219-015-9294-2.

[52] Micah D. J. Peters, Christina Godfrey, Patricia McInerney, Zachary Munn, Andrea C. Tricco, and Hanan Khalil. 2024. Scoping reviews. In *JBI Manual for Evidence Synthesis*. Edoardo Aromataris, Craig Lockwood, Kylie Porritt, Bianca Pilla, and Zoe Jordan, (Eds.) JBI. ISBN: 9780648848820. doi:10.46658/JBIMES-24-09.

[53] Anika Radkowitsch, Michael Sailer, Ralf Schmidmaier, Martin R. Fischer, and Frank Fischer. 2021. Learning to diagnose collaboratively – Effects of adaptive collaboration scripts in agent-based medical simulations. *Learning and Instruction*, 75, 101487. doi:10.1016/j.learninstruc.2021.101487.

[54] Michael Sailer, Elisabeth Bauer, Riikka Hofmann, Jan Kiesewetter, Julia Glas, Iryna Gurevych, and Frank Fischer. 2023. Adaptive feedback from artificial neural networks facilitates pre-service teachers' diagnostic reasoning in simulation-based learning. *Learning and Instruction*, 83, 101620. doi:10.1016/j.learninstruc.2022.101620.

[55] Bernhard Schmitz. 2006. Advantages of studying processes in educational research. *Learning and Instruction*, 16, 5, 433–449. doi:10.1016/j.learninstruc.2006.09.004.

[56] Viktor Shynkarenko and Oleksandr Zhevaho. 2020. Development of a toolkit for analyzing software debugging processes using the constructive approach. *Eastern-European Journal of Enterprise Technologies*, 5, 2 (107), 29–38. doi:10.15587/1729-4061.2020.215090.

[57] Gerry Stahl. 2005. Group cognition in computer–assisted collaborative learning. *Journal of Computer Assisted Learning*, 21, 2, 79–90. doi:10.1111/j.1365-2729.2005.00115.x.

[58] John Sweller, Jeroen J. G. van Merriënboer, and Fred Paas. 2019. Cognitive Architecture and Instructional Design: 20 Years Later. *Educational Psychology Review*, 31, 2, 261–292. doi:10.1007/s10648-019-09465-5.

[59] Andrea C. Tricco et al. 2018. PRISMA Extension for Scoping Reviews (PRISMA-ScR): Checklist and Explanation. eng. *Annals of internal medicine*, 169, 7, 467–473. eprint: 30178033. doi:10.7326/M18-0850.

[60] Iris Vessey. 1985. Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies*, 23, 5, 459–494. doi:10.1016/S0020-7373(85)80054-7.

[61] Noreen M. Webb, Philip Ender, and Scott Lewis. 1986. Problem-Solving Strategies and Group Processes in Small Groups Learning Computer Programming. *American Educational Research Journal*, 23, 2, 243–261. doi:10.3102/00028312023002243.

[62] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. 2023. A Think-Aloud Study of Novice Debugging. *ACM Transactions on Computing Education*, 23, 2, 1–38. doi:10.1145/3589004.

[63] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. 2021. Analysis of a Process for Introductory Debugging. In *Proceedings of the 23rd Australasian Computing Education Conference*. ACE '21: Australasian Computing Education Conference (Virtual SA Australia). Claudia Szabo and Judy Sheard, (Eds.) ACM, New York, NY, USA, 11–20. ISBN: 9781450389761. doi:10.1145/3441636.3442300.

[64] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE '14: 18th International Conference on Evaluation and Assessment in Software Engineering (London England United Kingdom). Martin Shepperd, Tracy Hall, and Ingunn Myrtveit, (Eds.) ACM, New York, NY, USA, 1–10. ISBN: 9781450324762. doi:10.1145/2601248.2601268.

[65] Wei Yan, Feiya Luo, Maya Israel, Ruohan Liu, and Latoya T. Chandler. 2025. How Do Elementary Students Apply Debugging Strategies in a Block-Based Programming Environment? *Education Sciences*, 15, 3, 292. doi:10.3390/educsci15030292.

[66] Stephanie Yang, Miles Baird, Eleanor O'Rourke, Karen Brennan, and Bertrand Schneider. 2024. Decoding Debugging Instruction: A Systematic Literature Review of Debugging Interventions. *ACM Transactions on Computing Education*. doi:10.1145/3690652.

[67] Andreas Zeller. 2009. *Why programs fail. A guide to systematic debugging*. eng. (Second Edition ed.). Morgan Kaufmann an imprint of Elsevier and dpunkt.verlag, Amsterdam et al. 400 pp. ISBN: 978-0-12-374515-6.

[68] Yingbin Zhang, Luc Paquette, Juan D. Pinto, Qianhui Liu, and Aysa Xuemo Fan. 2023. Combining latent profile analysis and programming traces to understand novices' differences in debugging. *Education and Information Technologies*, 28, 4, 4673–4701. doi:10.1007/s10639-022-11343-7.