Current Status and Perspectives of Debugging in the K12 Classroom: A Qualitative Study

Tilman Michaeli
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
tilman.michaeli@fau.de

Ralf Romeike
Freie Universität Berlin
Berlin, Germany
ralf.romeike@fu-berlin.de

***Abstract* – Self-reliance in debugging is both an important skill and a major challenge in learning to program. Debugging is distinct from general programming skills and needs to be taught explicitly. Nevertheless, when it comes to teaching and learning debugging, there are surprisingly few studies and results. The aim of this qualitative study is to investigate how students and teachers cope with errors in the K12 classroom, which debugging skills are conveyed, and why teachers teach or do not teach certain debugging skills. Therefore, in a first step, we identify skills considered relevant for debugging by applying desk research. We particularly focus on skills considered relevant for novices. Building upon this, we analyze 12 interviews of German high-school teachers using structured qualitative content analysis. The results show that especially weaker students are often helpless and apply a trial-and-error approach for coping with programming errors. It turns out that compile-time errors pose a big hurdle for many students. Teachers are mostly rushing from one student PC to the other, trying to help. Regarding the teaching of debugging skills, teachers focus on heuristics for common bugs as well as some debugging strategies. No systematic process on how to tackle and cope with errors is conveyed by teachers. Furthermore, they do not employ explicit teaching lessons on debugging. Overall, teachers lack a systematic approach for teaching debugging, as there are only insufficient concepts and materials.**

Keywords—debugging, K12, computer science education, qualitative content analysis, teacher perspectives

I. Introduction

Learning to program requires a variety of competences and poses a major challenge in computer science education. Students do not only need to grasp certain programming concepts, but they also need to persevere and find solutions in the case of errors. Systematically checking programs for errors, finding and fixing them is an essential competence for professional developers, who spend 20 to 40 percent of their working time on it [1]. Ultimately, a high degree of debugging knowledge leads to the acquisition of a systematic and planned troubleshooting-approach instead of trial-and-error [2]. On the other hand, it is well known, that programming novices have major problems dealing with errors. This poses a significant obstacle when learning to code. Furthermore, debugging is an approach discussed in the context of computational thinking [3]. Accordingly, debugging has been considered increasingly important in recent curricula such as the British computing curriculum. In Germany, where this study is situated, few computer science curricula explicitly include the term "debugging", while almost all contain the skill of "finding and fixing errors".

However, there is a lack of approaches, teaching materials and studies on how to teach and integrate debugging concepts effectively into K12 classrooms. To eventually develop suitable approaches, best practices, and materials for the classroom, we have to incorporate teachers' existing experience as well as their personal perspectives towards debugging. As teachers are confronted with students' programming errors on an everyday basis, we want to investigate their approaches and best practices.

To this end, the aim of this qualitative study is to investigate how students and teachers cope with errors in the classroom, which debugging skills are conveyed, and why teachers teach or do not teach certain debugging skills. Accordingly, we first need to identify the skills considered relevant for debugging. Building upon this, we can analyze which concepts and strategies for dealing with errors are applied and/or conveyed in the classroom.

Therefore, the following research questions are addressed:

- **(RQ0)** Which skills are considered relevant for debugging in literature? Which skills are considered relevant for novices?
- **(RQ1)** How do teachers and students cope with programming errors in the classroom?
- **(RQ2)** Which skills regarding debugging are taught in classrooms? When and how is debugging taught?
- **(RQ3)** What is the motivation of teachers to (not) teach debugging skills?

The paper is structured as follows: In section 2, related research work regarding relevant debugging skills and approaches to debugging in the classroom is discussed. In section 3, we outline the methodology of our study, which consists of two parts: desk research is used to address RQ0. Building upon this, a category system is developed and constitutes the basis for the analysis of how students and teachers cope with errors in the K12 classroom (RQ1), which debugging skills are conveyed (RQ2), and why teachers teach or do not teach certain debugging skills (RQ3) by applying a structured qualitative content analysis. This section is followed by the presentation of our results for both the desk research and the qualitative study in section 4. In section 5, the results are discussed and in section 6, we present our conclusions.

II. Related Work and Theoretical Background

A. Skills that help with Debugging

Debugging is the process of finding and fixing errors. Debugging skills are distinct from general programming skills, as [5], [6] or [8] show. They find that, while good debuggers are typically good programmers, persons with high proficiency in programming are not necessarily

proficient debuggers. However, high proficiency in debugging is a necessity for becoming a good software developer [1]. This leads to the question: what constitutes a good debugger?

Ducasse and Emde [7] give a classification of knowledge needed for debugging. Their results are based on an analysis of automatic debugging systems. They conclude that except for knowledge about the actual and intended program, the programming language and general programming expertise, knowledge about bugs and debugging techniques is necessary. However, not all types of knowledge are required in every debugging situation. These findings are confirmed by Ahmadzadeh [8].

One approach to analyze relevant debugging skills is to study and compare debugging experts and novices: Vessey [9], Gugerty and Olson [10], and Nanja and Cook [11] examine those differences by observing the debugging behavior of students or professionals. Among the main results are differences in program comprehension and understanding: experts are significantly faster in getting an overview over the program. Furthermore, experts formulate better hypotheses and are more flexible in their overall approaches and strategies. Novices use similar strategies but apply them inefficiently. However, all studies observe the debugging performance for given – and therefore foreign – code. Accordingly, comprehension poses a far more important part of the debugging process. Allwood [12] emphasizes the difference between debugging one's own programs and debugging other people's programs. Katz and Anderson [13] also show that the employed strategies vary based on whether the debugger is also the author of the code, or not. Fitzgerald et al. [6] investigate novices and find that the efficiency of applying debugging strategies, e.g. where *printf*-statements are placed and what they print, is crucial to debugging performance and distinguishes strong debuggers from weak ones. This is also reflected by McCauley et al. [4] in their extensive review of literature up to 2008.

The debugging process varies significantly depending on the type of the underlying error, in particular, due to the difference in available information. There is a large number of categorizations for error types (see e.g. [14], [15], [16], [17]). Typical distinctions are syntactic (mistakes in spelling or punctuation), semantic (mistakes regarding the meaning of the code) and logical errors (arising from a wrong approach or a misinterpreted specification), construct-related (in the sense of programming language specific constructs) and non-construct-related errors, or the distinction between compile-time and runtime errors.

Regarding error frequency for novices, it is hard to make quantitative statements, as studies either focus on only some specific error types (such as errors detectable by the compiler), lack an appropriate sample size or analyze only the final state of the programs (and no interim versions). Hall et al. [18] find that logical errors are most common, although their findings are based on a limited data set. Spohrer and Soloway [19] also state that non-construct-related errors are made much more frequently than construct-related errors, contrary to popular belief. Altadmri and Brown give an overview of bugs commonly encountered by novices, based on the analysis of the BlueJ-Blackbox data set. They conclude, that semantic errors are made more often than syntax errors, at least after a certain level of programming proficiency has been reached [20]. Regarding error severity – measured by the time needed to fix a specific error – syntax errors are fixed very quickly. Students find it harder to find errors not identified by the compiler. These errors take significantly longer to be found and fixed [20] [6].

B. Teaching Debugging

Murphy et al. [21], like Kessler and Anderson [22], argue that debugging techniques such as testing and tracing should be taught, as well as heuristics and patterns that help to apply these techniques. Nevertheless, when it comes to teaching debugging, there are surprisingly few studies and results; this is true in both academic (university) settings and K12:

1) University Settings

Katz and Anderson [13] trained students to use different debugging approaches (forward-reasoning, backward-reasoning) for debugging LISP programs. Student groups were first guided to use one of these approaches. Afterward, they were free to use whichever strategy meets their needs. In the end, the subjects continued to predominantly use the approach in which they were trained. Otherwise, hardly any differences were found between the approaches.

Allwood and Björhag [23] provided written debugging hints to an experimental group of undergraduate students. While the number of bugs did not differ between experimental and control group, the number of eliminated bugs (especially semantic and logical) was significantly higher. No difference was found in the strategies students tended to employ; this led to the conclusion that the differences must lie on a higher level: whether the approach was systematic, or not.

Chmiel and Loui [24] developed activities for a university course to foster the debugging skills of students. The activities were carried out – partly on a voluntary basis (debugging tasks, debugging diaries), partly mandatory (development diaries) – over the course of a semester. As the semester progressed, students that completed the voluntary debugging tasks needed significantly less time to debug their programs. However, this correlation was not reflected in the final exam results. Contrary to the expectation, the results were only slightly better.

Böttcher et al. [25] trained the debugging skills of students by introducing a systematic debugging approach, as well as the use of the debugger, in an explicit teaching lesson. The Wolf Fence algorithm (binary search) is used as an analogy and conveyed through written instructions. The lecturer made the debugging procedure explicit in a live coding demonstration, while a lab exercise included debugging tasks. The evaluation showed that only a few students implemented the binary search as demanded, and quickly returned to an unsystematic "poking around" and "visual diagnosis" behavior.

2) K12 Settings

Carver and Risinger [2] conveyed a debugging process with Logo, yielding promising results. 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. Results (without control group) showed a change from brute-force to a focused-search approach when searching for bugs. Furthermore, significantly less time was needed for finding errors. The students formulated more hypotheses before

trying out the code, paid more attention to the control flow, made fewer code changes (especially in bug-free places) and produced a lower number of new bugs.

In conclusion, we find that debugging skills are distinct from general programming skills. One approach to investigate skills that are considered relevant for debugging is the comparison of novices and experts. With regard to explicit teaching of debugging, there are unexpectedly few studies. Nevertheless, they provide a further source for debugging skills considered relevant, as well as for insights into experiences with the respective teaching approaches.

III. Methodology

The aim of this qualitative study is to investigate how students and teachers cope with errors in the classroom, which debugging skills are conveyed, and why teachers teach or do not teach certain debugging skills. To this end, we first need to identify the skills considered relevant for debugging. Building upon this, we can analyze which concepts and strategies for dealing with errors are applied and/or conveyed in the classroom. Therefore, this study consists of two parts: After identifying relevant debugging skills using desk research (RQ0), the question of how and why they are applied in the classroom is addressed using interview data (RQ1-3).

A. Desk Research addressing RQ0

To address RQ0, desk research was carried out, which resulted in skills considered relevant for debugging. For this purpose, relevant libraries and journals (ACM Digital Library, IEEE Digital Library, Google Scholar) were systematically searched by examining the occurrences of respective keywords ("debugging", "debugging strategies", "debugging education", "debugging competences", "debugging skills") in documents. As a further selection criterion, the documents had to have at least four pages. This was accompanied by a thorough analysis of different programming textbooks (student textbooks used in German school curricula (10), standard textbooks (17) and books with a focus on debugging (4)). The resulting documents were analyzed for statements and results regarding the following aspects:

1) What skills are applied by debugging experts and, how do they differ from novices?

2) Which debugging skills are taught or considered relevant?

Novices and experts must be considered separately. Accordingly, in our analysis, we distinguish two groups of skills: those considered relevant only for intermediates or professional developers, and those relevant for programming novices, particularly in K12 or entry-level university.

As discussed in section 2, debugging skills are conceptually different from general programming skills. Therefore, this analysis focuses particularly on debugging skills – in contrast to skills considered general programming skills. For that reason, we omitted such skills as program comprehension and programming concept knowledge, although they are evidently necessary requirements for the debugging process.

B. Interview study addressing RQ1-3

For the analysis of how students and teachers cope with errors in the classroom, which debugging skills are conveyed, and why teachers teach or do not teach certain debugging skills, we conducted semi-structured interviews with twelve outstanding high school teachers from different regions of Germany (leading to a wide range of different requirements, curricula, etc.). Through the qualitative examination of the teacher's perspective, we can gain a deeper insight into the teaching practice. Therefore, we can not only investigate applied debugging skills but also evaluate the experience teachers gathered in doing so as well as their motivation.

In doing so, we want to determine best practices and suitable approaches. Therefore, our sample group contains only well-experienced teachers, that are either involved in teacher training and/or have solid connections to universities.

Most of the teachers teach object-oriented programming and almost exclusively use Java as their programming language. Therefore, we only considered the teachers' experience using text-based programming languages.

For the analysis of the data collected in interviews, we applied a structured content analysis approach according to Mayring [26]. The software MaxQDA was used for the actual coding. We deductively developed a category system, building upon the results of RQ0. To avoid neglecting important aspects due to previously-defined categories, inductive additions were allowed. The transcribed interviews form the basis of the evaluation. The related text passages of the interviews serve as units of analysis. By classifying text passages, individual depictions are generalized across all cases. As for intercoder reliability, a second researcher coded the text snippets for two of the transcribed interviews (20 percent of the codings) as well.

IV. Results

A. RQ0: Which skills are considered relevant for debugging in literature? Which skills are considered relevant for novices?

The desk research resulted in four debugging skills considered relevant for novices, which will be explained in the following.

1) Applying a systematic debugging process

A systematic debugging process – sometimes referred to as "strategy" – is the high-level systematic pursuit of a plan to find and correct errors. With respect to professionals, there is a large number of debugging process models that share the following common aspects (such as [13], [27], [9], [28], [29], [30]):

- Testing and making the program fail
- Gaining an overview of the program (if necessary)
- Formulating a hypothesis
- Experimentally verifying the hypothesis
- If necessary, repeated refinement of the hypothesis
- Correcting the error
- Testing the program again

Zeller [28] labels this approach of generating and refining hypotheses as scientific debugging – derived from the scientific method – treating an error therefore as a "natural phenomenon". In a recent study on the status quo of debugging in industry, all participating developers described their debugging approach as being similar to the (simplified) scientific method [1].

Adaptations for novices can be found (c.f. [2], [25], or [31]). Overall, these adaptations simplify the process. Therefore, they omit steps such as tracking the bug in a database or adapting regression tests.

2) Applying debugging strategies

In contrast to a debugging process, debugging strategies – sometimes referred to as "tactics" or "techniques" – are lower level practices, supporting the steps of finding and refining hypotheses. Examples are tracing the program flow using *printf*-debugging or the debugger, slicing or forcing the execution of a specific case. One of the key differences separating experts from novices is the effectiveness debugging strategies are applied with [4].

There is a wide variety of strategies, see for example [30], [1], [32]. Murphy et al. [21] give an overview of strategies commonly employed by novices. In entry-level programming textbooks, strategies such as *printf*-debugging, using the debugger, trace tables, and slicing can be found.

3) Applying heuristics and patterns for common bugs

Developers often apply a shortened version of the systematic debugging process, including formulating and testing hypotheses: From their experience, they know typical errors and their possible causes. To support this "learning from past errors", many professional developers keep a debugging diary or debugging log, which they use to document their debugging experience. This process helps to build a catalog of heuristics that supports the removal of similar bugs in the future [1].

4) Usage of Tools

The use of professional debugging tools such as automatic fault localization approaches and back-in-time-debuggers strongly depends on the application domain. Regarding tools considered relevant for programming novices we only found the use of standard debuggers – which falls into the category debugging strategy as well. In addition, utilizing IDE feedback was sometimes mentioned.

Building upon these results, we deductively developed a category system for the interview analysis (see Table I). The first category addresses the teachers' experiences regarding their and students' coping with programming errors in the classroom, including common student errors and the teacher-student interaction. The next category covers teachers' statements about approaches to teaching debugging: which debugging content (in the sense of the respective skills from our desk research), and how and when it is taught. Within the last category, teacher's motivation regarding teaching debugging is addressed.

B. RQ1: How do teachers and students cope with programming errors in the classroom?

1) Students' Reactions to Errors

Finding 1: Students' reactions to errors vary depending on the teacher.

The observations of the teachers regarding the students' reactions to errors differ. Some teachers report that the majority of the students report problems to the teacher immediately. Others report that they first try to solve errors on their own, or with their neighbor. Sometimes, however, this behavior might merely be a result of the teachers' unavailability, possibly because they are busy helping other students:

"First of all, they have to try it on their own because most of the time I am busy helping somewhere else."

TABLE I. CATEGORY SYSTEM FOR STRUCTURED CONTENT ANALYSIS

Category	Subcategory	Exemplary Coding
Coping with errors in the classroom	Information about students' reactions to errors	The motivated and enthusiastic students try to eliminate errors on their own.
	Information about teachers' reactions to errors in student code	I notice when a student puts up his hand. Then, I go there and try to help him.
	Characteristics of student-teacher interaction	Students just say: "there was an error message, I don't know what it said".
	Information about common student's errors	Typical errors are, for example, that they try to call a method without creating the respective object beforehand.
Debugging skills taught in the classroom	**Statements about teaching a systematic debugging process**	No, I do not introduce something like that.
	Statements about teaching debugging strategies	Then I point out to them: "just insert a line which prints exactly what you want to know".
	Statements about teaching heuristics and patterns for common bugs	Then I try to explain where the error typically comes from.
	Statements about teaching the usage of tools.	I introduce the debugger as a tool.
	Ways of teaching debugging, such as *on an individual basis* or in *explicit debugging lessons*	If it is a common error, then I always address it in front of the whole class, if it is an individual problem, then only with the respective student.
	Statements regarding the point of time when debugging is taught, such as at *the beginning of the course* or *on demand*	Usually when we talk about arrays, I introduce the debugger.
	Kinds of activities used for teaching and supporting debugging	I'm handing out erroneous source code for practicing.
Motivation of teachers to (not) teach debugging	Statements about why teachers include debugging contents in their teaching	In particular, the helplessness and frustration of students.
	Statements about why teachers neglect debugging contents in their teaching	I just have to keep it short because the curricular topics have priority.
	Statements of teachers as to whether they refined their teaching regarding debugging over the course of their professional experience	What has developed over time is my experience: Is this really an individual problem, or is it a problem that quite a few students will struggle with?

Finding 2: "Good" students have fewer problems with debugging, while "weak" students are rather helpless.

If students try to fix errors on their own, teachers distinguish two groups of students: "good" students who need little help, and "weaker" students. The latter are frequently overwhelmed and apply an unsystematic trial-and-error approach. One teacher sums up his experience as follows:

> *"Those who can program recognize the error and fix it, and those who can't program just try something until the error message is gone [...] they just keep adding int's or semicolons until this error message doesn't occur anymore."*

Finding 3: Compile-time errors pose a major hurdle for students, even after some programming experience.

Furthermore, the teachers state that simple syntax errors (such as missing semicolons or parentheses) are no longer a problem after a few weeks. However, many teachers agree that other types of compile-time errors remain a big problem. When asked about logical errors, one teacher responds:

> *"In grade 10, it is less of a problem, because most students don't even succeed in making their program run completely within the time frame of a lesson."*

One teacher describes problems resulting from ambiguous (Java-) error messages that appear in the wrong place – at least from the student's point of view.

> *"'reached end of file while parsing' or similar error messages are not obvious for the students, because the error message points to a different location."*

Finding 4: Students ask unspecific questions.

Overall, students ask predominantly unspecific questions when they eventually call the teacher for assistance.

> *"The majority of students look horrified, put their hands up and say 'that's red, there's a mistake', and expect the teacher to present the solution to them"*

2) Teachers' Reactions:

Finding 5: Teachers are mostly rushing from one student PC to the other, trying to help.

In general, teachers noticed a great amount of helplessness and frustration when dealing with errors, so that the teacher actually ends up rushing from one student PC to the other, explaining the error and giving hints for troubleshooting.

> *"With most error messages, when you enter them in Google, you find the solution, but relatively few dare or want to do that. They always like to have the teacher next to them."*

Finding 6: When helping students individually, teachers predominantly address underlying misconceptions.

In the small time frames allotted to each student, the primary goal usually is to eliminate underlying misconceptions.

> *"Depending on how much time you have for the individual, you either really try to sit down and tell them: 'Read this, what does it mean? What did you do differently when there was no error?' But if you have a lot of requests, [...] then you quickly tell them 'there's a semicolon missing', 'there's a small s', or 'you forgot the new'."*

Finding 7: There are some teachers that demand a great deal of autonomy in handling errors.

This does not apply to all teachers, however: there is a small number of teachers who follow the described approach only in the first few weeks and then demand independent thinking and research from the students. In these cases, before the teacher may be called for help, students are expected to make autonomous attempts at solving errors and consult classmates first. One teacher, for example, stated that one of his key principles is that each error message will be explained to the class only once:

> *"Such error messages are addressed once for all students, but only once. They know that each error message may be asked once, afterward I tell them 'no, we already had that, either you have noted it down or you to remember it, or someone in the group knows it, it is no longer my concern', I am persistent in that."*

> *"Otherwise, I would be the replacement for the compiler, always saying, 'remember, there is a semicolon missing'. Then they wouldn't even dare to press compile. No, they just have to learn how to deal with the errors."*

Finding 8: Teachers that demand autonomy report a better handling with errors and fewer problems with compile-time errors.

Interestingly enough, teachers following this course of instruction also tend to see compile-time errors as a lesser hurdle.

> *"The syntax things are overcome routinely and relatively quickly."*

One teacher reports more self-reliance, more specific questions, and overall better handling of errors compared to his teaching before employing this demanding-self-reliance approach.

> *"I was used to experiencing this situation quite often, 'yes there was an error message', 'what did it say?', 'I don't know, I got rid of it. The most important thing about the error message was the red cross at the top right, close it, get rid of it, and then call the teacher."*

C. RQ2: Which skills regarding debugging are taught in classrooms? When and how is debugging taught?

1) Skills taught in the Classroom

Finding 9: Teachers convey no **systematic debugging process.**

With regard to skills taught, the teachers reported a lack of systematic approaches. Evidently, the skills addressed strongly depend on the programming environments and languages used in class. None of the teachers teaches any kind of debugging model process that goes beyond "first of all, read the error message".

Finding 10: A variety of unsystematic **debugging strategies** are taught.

The majority of teachers gives at least one debugging strategy to the students, but – with some exceptions – these strategies are not systematically taught or practiced. A focus seems to be on tracing strategies. Therefore, the usage of the debugger, *printf*-debugging, or the use of other tool-specific features that help with tracing – see finding 12 – were among the most frequently-named strategies. Furthermore, internet research for the handling of error messages, commenting-out, basic slicing, and testing (especially testing by visual observation) were sometimes mentioned.

Finding 11: Teachers focus on **heuristic and patterns for common bugs**.

Nevertheless, the teachers' main focus is placed on the explanation of error messages and heuristics for dealing with typical errors, especially for runtime and compile-time errors.

> *"I try to clarify where the error typically occurs. Often the loop goes on for too long, you made your array too small or forgot that it starts at 0. Things like that."*

Finding 12: Usage of tools in the classroom is dominated by the debugger and further tool-specific features that help with tracing, although with mixed results.

Regarding the usage of tools, teachers report using the debugger – although almost exclusively in a didactically-reduced version (as found i.e. in BlueJ) – as well further tool-specific features, like the object inspector in Greenfoot and BlueJ. This object inspector enables displaying the values of an object's static and instance fields at any given time [33].

For the debugger teachers agree that it only seems to help the "good" students. This is also reflected in its usage: they are the only group that actively uses it.

> *"With the "good "students I also use the debugger [...], but that strategy is less helpful for the "weaker" students, [...] I notice that they don't have the courage to use it."*

One teacher reports that introducing the debugger did not work out. Consequently, for them, it remained a one-time experiment.

> *"I tried it once, [...] it required even more effort and caused confusion."*

2) When Debugging is Taught

With regard to the point in time when debugging is taught, we have distinguished between three procedures: at the beginning of the course, after a certain level of programming proficiency has been reached or on demand – when respective skills are needed.

Finding 13: Attempting to build knowledge predominantly in the beginning of the course does not benefit the students

Some teachers reported putting measures into place to try to prevent problems early on in the programming course. These measures could include differentiation of various error types, basic heuristics for the handling of specific errors (or error messages), or the introduction of a tool-specific debugger or object inspector. They reported, however, that the content and scaffolding materials conveyed in previous weeks saw little use, as one teacher states:

> *"The students think they understood it, and then they never look at it again."*

Finding 14: Most debugging skills are taught on demand.

A common example for this is the introduction of arrays. For this topic, the corresponding error messages (and their typical causes) or strategies, such as usage of the debugger, are commonly covered at two possible stages: during the general introduction of the topic, or in response to the first time an unauthorized access-to-memory error occurs.

> *"With arrays, this ArrayIndexOutOfBounds-Exception, when it comes up for the first time, I usually display the students' screen on the projector and tell them: 'listen people, each of you might encounter this error message in the next few weeks, that could be the problem behind it."*

3) How Debugging is Taught

Finding 15: Teachers tend not to employ explicit teaching lessons on debugging, but instead teach the relevant skills on an individual basis.

None of the teachers employ an explicit lesson aiming at debugging, other than the introduction of the debugger. If contents concern all students (e.g. the first occurrence of a specific error, introduction to the debugger, ...), these are addressed in front of the entire class. However, a large part of the support provided to students is individual in nature. The teachers also reported that learning occurred through observation in these individual support phases, e.g. how to use the strategy of commenting- out.

> *"It is often the case that I demonstrate it once. So it is more like 'learning by observation' than 'learning by instruction'."*

Concerning certain debugging-related activities, some of the teachers mentioned using debugging tasks. Debugging diaries or reviews have also been mentioned by a few teachers.

D. RQ3: What is the motivation of teachers to (not) teach debugging skills?

Finding 16: Teachers convey debugging skills mostly due to their experience of students' helplessness.

The primary reason given by teachers for the integration of debugging contents and skills is the students' perceived helplessness in dealing with programming errors. Some of the teachers reported a further development of their debugging content adapted to the typical errors encountered by students over the years. All teachers would like to integrate more debugging-related topics into their teaching.

Finding 17: The main reported factors to not include debugging in teaching are: lack of time, debugging not being an explicit topic in the curriculum, and missing concepts and materials.

Regarding the reasons why they do not teach more debugging in class, the teachers mainly reported a lack of time. This includes both lesson time and time for the preparation and creation of suitable concepts.

Another reason given is that debugging is not an explicitly-named content of the curriculum – despite it being a central step in programming:

> *"In the curricula, it is not included as an explicit topic, [...] and okay, I do programming, what do I need for programming, I need the programming concepts, I need the data structures, so these are topics which then end up in the lesson plan, and debugging is more related to the process, and therefore, seldom a subject of teaching."*

Therefore, teachers often "neglect" debugging in favor of content explicitly required by curricula. This may also cause a lack of awareness of the subject area. Furthermore, they argue that they do not know any adaptable concepts and require an approach to conveying more content in a suitable way. They claim that there is no material, not even in textbooks for educational settings:

> *"You can find something about databases, about programming, about all these fields of computer science, but [debugging] is rarely a topic on its own."*

Finding 18: An important source for teachers' debugging content knowledge is their own debugging behavior.

Four of the teachers explicitly stated that the strategies and support they provide to students are based on their own personal debugging behavior:

> *"I was considering, how did I do that back then at university?"*

V. Discussion

This qualitative study was designed to investigate how students and teachers cope with errors in the classroom, which debugging skills are conveyed, and why teachers teach or do not teach certain debugging skills. Therefore, in a first step, relevant debugging skills were identified in desk research. Building upon this, interview data were analyzed with a structured qualitative content analysis.

By deliberately selecting teachers who either cooperate with universities and current research or are involved in teacher training, the aim was to collect best practices and well suitable approaches. Therefore, these results are more likely to be the upper limit of what is done regarding debugging in classrooms.

Overall, our findings on relevant debugging skills in literature matched the teachers' statements on debugging skills they convey: teachers focus on a variety of debugging strategies (finding 10) as well as on heuristics for common bugs (finding 11) and the usage of tools (finding 12). However, this does not apply to the application of a systematic debugging process (finding 9). Thus, no coding could be assigned for this category, despite being an explicit point of inquiry. Teachers presumably apply a systematic approach when debugging. Possibly, this may only be a subconscious process, since teachers themselves might have learned to debug in an unstructured way – as most of professional software developers have [1]. Therefore, they might not consider such a systematic approach as content relevant for fostering debugging. However, explicitly teaching a systematic debugging process has promising results [2]. The data further indicates, that educators teach debugging strategies based on their own debugging behavior (finding 18): A teacher who relies on the debugger to find errors may predominantly teach proper debugger usage, whereas a teacher who rely on working with *printf*s may instead teach how to use those. This insight can be helpful in teacher training: introducing a systematic approach to debugging will potentially benefit future students.

Beside debugging expertise, general programming skills are often a prerequisite for dealing with errors, as [7] and [8] indicate. The teachers also emphasize that they want to prevent errors by successfully conveying programming concepts. This is in line with the implication for teaching and learning debugging McCauley et al. [4] draw from their literature review: combating misconceptions, as they pose a major source of errors. Therefore, errors in the classroom not only provide valuable learning opportunities for the students but are also indicators for the teachers as to whether the concepts have been successfully acquired. Potentially, teaching and practicing debugging might even pose a valuable opportunity to improve students' general programming skills: tracing and tracing strategies play an important role not only in the debugging process [34], but also for overall programming. By improving their tracing skills, students improve their model of notional machine and overall program understanding skills [35].

Another thesis that emerges from the data is that teachers who demand a high degree of autonomy tend to see compile-time errors as a lesser hurdle. They also observe more autonomy in troubleshooting (finding 8). Therefore, it seems promising to put a strong focus on self-reliance and supporting it in materials and concepts. The concept of "learned helplessness" [38] might play a role here, as there are no notable differences in the type of debugging strategies taught to students; however, it must be noted that two out of these three teachers employ (agile) project-based learning early on, which may have had a significant influence on their epistemological approaches to teaching. Though, we cannot

investigate this thesis further with the given data, but it offers an interesting perspective for future research.

The data further indicates, that compile-time errors represent a major hurdle for many students (finding 3). In order to appropriately address this hurdle, a systematic approach to properly deal with error messages is required – at least when teachers use traditional text-based programming, and not block- or frame-based approaches, where many of these errors are no longer possible [36]. For this reason, related skills should also be incorporated into a pedagogical approach. Merely relying on the conveyance of strategies such as tracing is not sufficient. This perspective differs from the predominant focus on debugging for already compiling programs at university level. Some even define debugging as starting at a runtime level [14]. One possible explanation for this difference can be the limited teaching time in the school: At the end of the lesson at the latest, the students receive the solutions for the exercises, no matter how many errors are still remaining in their program. Therefore, the students do not form any heuristics and experiences of how to deal with certain errors. At the same time, it must be noted that educators assess the errors frequently made by students poorly [37].

Regarding the teaching of debugging skills, teachers confirm a lack of materials (finding 17). The primary target group for interventions and materials are "average" to "weak" students, "good" students also cope with the status quo (finding 2, finding 12). The unsystematic trial-and-error approach that teachers reported for those "weaker" students is in line with literature [21].

VI. Conclusion

In summary, we found four primary debugging skills considered relevant for novices in our analysis: the application of a systematic high-level debugging process, the application of low-level debugging strategies, the application of heuristics and patterns for common errors, and the usage of tools.

Regarding how teachers and students cope with errors in the classroom, the teachers report that especially "weaker" students are often overwhelmed and helpless when dealing with errors. They often use a trial-and-error approach that is not very effective and show little self-sufficiency. Teachers often rush from one student PC to the other, trying to help students troubleshoot. It turned out that compile-time errors also pose a big hurdle for many students.

Concerning debugging skills conveyed in the K12 classroom, the results show that some strategies and heuristics, but no debugging process models – and therefore no systematic process on how to tackle and cope with errors – are taught. Other than the introduction of the debugger, none of the teachers employ an explicit lesson aiming at debugging. Teachers lack a systematic approach to teaching debugging: The data indicates that educators teach debugging based on their own debugging process, which they themselves have typically acquired in an unstructured way.

The main reason for teachers not to teach debugging skills is a lack in time – in lessons as well as for the preparation of suitable concepts and materials. Furthermore, debugging not being an explicit content of the curriculum and missing concepts and materials are reported by the teachers.

To help us create concepts and materials for the classroom, we can derive the following design principles from our results: The concepts and materials should:

- primarily target "weak" to "average" students' requirements,
- focus on self-reliance and supporting it,
- emphasize a high-level systematic debugging process,
- include approaches for coping with compile-time errors,
- and introduce debugging strategies (such as tracing strategies) and tools systematically.

In summary, this study offers deep insights into the current status of debugging in the K12 classroom and supports building a valuable foundation for addressing the lack of concepts and materials.

References

[1] *M. Perscheid, B. Siegmund, M. Taeumel, and R. Hirschfeld, Studying the advancement in debugging practice of professional software developers, Softw. Qual. J., vol. 25, no. 1, pp. 83-110, 2017.*

[2] *M. S. Carver and S. C. Risinger, Improving childrens debugging skills, in Empirical studies of programmers: Second workshop, 1987, pp. 147-171.*

[3] *A. Yadav, N. Zhou, C. Mayfield, S. Hambrusch, and J. T. Korb, "Introducing Computational Thinking in Education Courses," in Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education, 2011, pp. 465–470.*

[4] *R. McCauley et al., Debugging: a review of the literature from an educational perspective, Computer Science Education, vol. 18, no. 2, pp. 67-92, 2008.*

[5] *P. Romero, B. Du Boulay, R. Cox, R. Lutz, and S. Bryant, Debugging strategies and tactics in a multi-representation software environment, Int. J. Hum. Comput. Stud., vol. 65, no. 12, pp. 9921009, 2007.*

[6] *S. Fitzgerald, G. Lewandowski, R. McCauley, L. Murphy, B. Simon, L. Thomas and C. Zander, Debugging: finding, fixing and flailing, a multi- institutional study of novice debuggers, Computer Science Education, vol. 18, no. 2, pp. 93-116, 2008.*

[7] *M. Ducasse and A.-M. Emde, A review of automated debugging systems: knowledge, strategies and techniques, in Proceedings of the 10th international conference on Software engineering, 1988, pp. 162-171.*

[8] *M. Ahmadzadeh, D. Elliman and C. Higgins, An analysis of patterns of debugging among novice computer science students, ACM SIGCSE Bulletin, vol. 37, no. 3, p. 84, 2005.*

[9] *I. Vessey, Expertise in Debugging Computer Programs: Situation-Based versus Model-Based Problem Solving, Int. Conf. Inf. Syst., p. 18, 1985.*

[10] *L. Gugerty and G. Olson, Debugging by skilled and novice programmers, ACM SIGCHI Bull., vol. 17, no. 4, pp. 171-174, 1986.*

[11] *M. Nanja and C. R. Cook, Empirical Studies of Programmers: Second Workshop, G. M. Olson, S. Sheppard, and E. Soloway, Eds. Norwood, NJ, USA: Ablex Publishing Corp., 1987, pp. 172-184.*

[12] *C. M. Allwood and C.-G. Björhag, Novices debugging when programming in Pascal, Int. J. Man. Mach. Stud., vol. 33, no. 6, pp. 707-724, 1990.*

[13] *I. Katz and J. Anderson, Debugging: An Analysis of Bug-Location Strategies, Human-Computer Interaction, vol. 3, no. 4, pp. 351-399, 1987.*

[14] *A. J. Ko and B. A. Myers, A framework and methodology for studying the causes of software errors in programming systems, J. Vis. Lang. Comput., vol. 16, no. 12, pp. 4184, 2005.*

[15] *M. Hristova, A. Misra, M. Rutter, R. Mercuri, and B. Mawr, Identifying and Correcting Java Programming Errors for Introductory Computer Science Students, 2003.*

[16] *W. L. Johnson, E. Soloway, B. Cutler, and S. Draper, Bug catalogue: I. Yale University Press, 1983.*

[17] *J. G. Spohrer and E. Soloway, Analyzing the high frequency bugs in novice programs, in Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers, 1986, pp. 230251.*

[18] *M. Hall, K. Laughter, J. Brown, C. Day, C. Thatcher, and R. Bryce, An empirical study of programming bugs in CS1, CS2, and CS3 homework submissions, J. Comput. Sci. Coll., vol. 28, no. 2, pp. 8794, 2012.*

[19] *J. C. Spohrer and E. Soloway, Novice mistakes: Are the folk wisdoms correct?, Commun. ACM, vol. 29, no. 7, pp. 624632, 1986.*

[20] *A. Altadmri and N. C. C. Brown, 37 Million Compilations, Proc. 46th ACM Tech. Symp. Comput. Sci. Educ. - SIGCSE 15, pp. 522527, 2015.*

[21] *L. Murphy, G. Lewandowski, R. McCauley, B. Simon, L. Thomas and C. Zander, Debugging: the good, the bad, and the quirky a qualitative analysis of novices strategies, ACM SIGCSE Bulletin, vol. 40, no. 1, p. 163, 2008.*

[22] *C. M. Kessler and J. R. Anderson, A model of novice debugging in LISP, in Proceedings of the First Workshop on Empirical Studies of Programmers, 1986, pp. 198-212.*

[23] *C. Allwood and C. Björhag, Training of Pascal novices' error handling ability, Acta Psychologica, vol. 78, no. 1-3, pp. 137-150, 1991.*

[24] *R. Chmiel and M. C. Loui, Debugging: from Novice to Expert, Proc. 35th SIGCSE Tech. Symp. Comput. Sci. Educ. - SIGCSE 04, vol. 36, no. 1, p. 17, 2004.*

[25] *A. Böttcher, V. Thurner, K. Schlierkamp, and D. Zehetmeier, Debugging students debugging process, Proc. - Front. Educ. Conf. FIE, vol. 2016Novem, 2016.*

[26] *P. Mayring, Qualitative content analysis: theoretical foundation, basic procedures and software solution, 2014.*

[27] *J. D. Gould, Some psychological evidence on how people debug computer programs, Int. J. Man. Mach. Stud., vol. 7, no. 2, pp. 151-182, 1975.*

[28] *A. Zeller, Why Programs Fail: A Guide to Systematic Debugging. 2005.*

[29] *D. J. Gilmore, Models of debugging, Acta Psychol. (Amst)., vol. 78, no. 13, pp. 151172, 1991.*

[30] *D. Spenellis, Modern Debugging : The Art of Finding a Needle in a Haystack, Commun. ACM, no. November 2018, pp. 124134, 2018.*

[31] *A. Sipitakiat and N. Nusen, Robo-Blocks: designing debugging abilities in a tangible programming system for early primary school children, Proc. 11th Int. Conf. Interact. Des. Child. - IDC 12, no. December, p. 98, 2012.*

[32] *R. C. Metzger, Debugging by thinking: A multidisciplinary approach. Digital Press, 2004.*

[33] *M. Kölling, B. Quig, A. Patterson, and J. Rosenberg, "The BlueJ system and its pedagogy," Comput. Sci. Educ., vol. 13, no. 4, pp. 249–268, 2003.*

[34] *D. N. Perkins and F. Martin, "Fragile knowledge and neglected strategies in novice programmers," in first workshop on empirical studies of programmers on Empirical studies of programmers, 1986, pp. 213–229.*

[35] *A. Venables, G. Tan, and R. Lister, "A closer look at tracing, explaining and code writing skills in the novice programmer," Proc. fifth Int. Work. Comput. Educ. Res. Work. - ICER '09, no. 2008, p. 117, 2009.*

[36] *A. Altadmri, M. Kölling, and N. C. C. Brown, The Cost of Syntax and How to Avoid It: Text versus Frame-Based Editing, Proc. - Int. Comput. Softw. Appl. Conf., vol. 1, pp. 748753, 2016.*

[37] *N. C. C. Brown and A. Altadmri, Investigating novice programming mistakes: Educator Beliefs vs. Student Data Proc. tenth Annu. Conf. Int. Comput. Educ. Res. - ICER 14, pp. 4350, 2014.*

[38] *C. Peterson, S. F. Maier, and M. E. P. Seligman, Learned helplessness: A theory for the age of personal control. Theory for the Age of Personal, 1993.*