THE INVESTIGATION OF NOVICE PROGRAMMERS' DEBUGGING BEHAVIORS TO INFORM INTELLIGENT E-LEARNING ENVIRONMENTS: A CASE STUDY

Gamze TURKMEN
ORCID: 0000-0002-4695-9159
Faculty of Education
Middle East Technical University
Ankara, TURKEY

Sonay CANER
ORCID: 0000-0002-0381-2798
Faculty of Education
Middle East Technical University
Ankara, TURKEY

Received: 25/06/2019 **Accepted:** 11/01/2020

ABSTRACT

This study aims to provide a comprehensive and in-depth investigation of the debugging process in programming teaching in terms of cognitive and metacognitive aspects, based on programming students who demonstrate low, medium, and high programming performance and to propose instructional strategies for scaffolding novice learners in an effective way. Data were collected from 41 novice programming students from C++ and Python programming language courses of the same instructor in the scope of case study methodology, and data instruments included paper-based programming questions. The questions were framed under three categories as questions' difficulty levels (low, moderate and high), error types (syntax and logic), and question types (if-else and while). As having three categories, a total of 12 different data (3x2x2) were taken from each student, which means 492 data rows were evaluated in the study. Chi-square test results revealed that while error detection and correction are significantly high in low difficulty level questions, error detection and error correction attempts for logic errors were substantially higher compared to syntax errors. Further analysis conducted for paper-based markings that were used by students throughout their error detection, correction, and completion attempts. Chi-square test results revealed significant relationships between marking availability and error types, as well as difficulty levels. Results were discussed for both traditional learning and e-learning environments in terms of what kind of educational implications and strategies can be outlined by data for increasing the effectiveness of programming education for novice learners.

Keywords: Programming instruction, debugging, novice programmers, correlational study, intelligent e-learning environments

INTRODUCTION

Programming is a complex process where there are many sub-stages, and each sub-stage requires different knowledge and cognitive activities (Ambrosio, Costa, Almeida, Franco, & Macedo, 2011). This complex process can be a challenging and disappointing situation for novice learners (Fitzgerald et al., 2008). Studies conducted in the field of cognitive psychology stated that programming is beyond low-level mental activities (Weinberg and Shulman, 1974). Besides, studies on programming claim that debugging in programming are one of the critical indicators of one's programming skills (Ahmadzadeh, Elliman, & Higgings, 2005; Fitzgerald et al., 2008; Ko & Myers, 2003; Nivala, Hauser, Mottok & Gruber, 2016). Several studies have described the debugging process as a complex cognitive activity that requires different types of information and a complete mapping of the programming process (McGill & Volet, 1997; Soloway & Ehrlich, 1984). That is, comprehending programming is a complicated process for students (Lin et al., 2016). Researchers conducted various studies on programming. The common point of all these studies is to examine the pedagogical tools and methods that can make the programming skills of the novice programmers more effective (Brooks, 1977; Lin et al., 2016; Renumol, Janakiram, & Jayapraksh, 2010; Xu, & Rajlich, 2004). One of these studies emphasized the importance of the teacher's appropriate guidance in acquiring debugging skills and stated that students should be encouraged to develop the cognitive skills necessary for debugging. In addition, it emphasized the importance of developing adaptive teaching strategies with educational media. However, to develop these strategies with media, it is necessary to study the cognitive processes during debugging processes (Lin et al., 2016). Lin et al. (2016) reviewed the cognitive processes of novice learners when they are engaging in debugging activities. The study was based on low and high-performance programming skills during debugging. They suggested that students should be divided into groups according to their programming success, providing an environment where they could think aloud and creating a note-taking the field on the screen for the debugging process. Apart from examining programming in terms of cognitive processes, some studies have made suggestions to make these processes more productive, and some have examined the effectiveness of instructional tools that have been developed based on various instructional strategies. While a number of studies emphasized the effectiveness of the developed instructional tools (Salleh, Shukur, & Judi, 2018; Stachel et al., 2013), others stated that the tool they developed was not effective (Denny, Luxton-Reilly, & Carpenter, 2014).

Debugging in programming is the process by which programmers can determine the potential errors in the source code and resolve these errors. There are three types of errors, regardless of the programming language, which are syntax errors, system errors, and logic errors. Syntax errors are caused by incorrect use of the programming language syntax or violations of syntax rules, which are the errors that can be captured easily by compilers. System errors are errors that can be detected when the program is running, and they are mostly related to numerical calculations. Lastly, logic (semantic) errors, which are caused by incorrect translation of the algorithm, is the type of error that compilers cannot determine. However, it is possible for the programmers to carefully detect the logic errors in the source code in their minds (Savitch, 2012). Numerous studies on programming have focused on error types. Although logic errors are perceived more challenging to detect and correct, various studies have indicated that the detection and correction of syntax errors are not as easy as it is thought (Denny et al., 2014; Hristova, Misra, Rutter, & Mercuri, 2003; McCauley et al., 2008). One of these studies claimed that the error messages given by the compiler should be more meaningful to be understood by students (Hristova et al., 2003).

Denny et al., (2011) claimed students at different levels of programming knowledge experienced difficulty in debugging syntax errors. They observed that even the top quarter students in the classroom had more difficulty than they expected in correcting syntax errors. Literature suggested enhanced compiler feedback as a solution to this issue. Denny et al. (2014) developed an enhanced feedback module for syntax errors; however, empirical study results revealed the ineffectiveness of the module and offered follow-up researchers for investigation of compiler feedbacks. On the other hand, numerous studies conducted on logic errors. Based on the literature review of Hristova et al. (2003), conducted a study on Java programming language errors. They analyzed both their own programming experience and faculty's, teaching assistants', and students' experienced and revealed six common logic errors. These are (p. 155):

1. Improper casting,
2. Invoking a non-void method in a statement that requires a return value,
3. Flow reaches the end of the non-void method,
4. Methods with parameters: confusion between declaring parameters of a method and passing parameters in a method invocation,
5. Incompatibility between the declared return type of a technique and in its invocation,
6. Class declared abstract because of the missing function.

Debugging has a critical role in programming. Several studies stated that students who are competent in debugging are also skilled in programming (Ahmadzadeh, Elliman, & Higgings, 2005; Fitzgerald et al., 2008). While some studies claimed that debugging is one of the two essential skills that show competence in programming (Nivala, Hauser, Mottok, & Gruber, 2016), some have taken this claim to a higher level and argued that debugging is at the highest importance in programming (Ko & Myers, 2003). By the fact that this importance, debugging can be complicated, and novice learners need to synthesize their new knowledge and skills with previous ones. Moreover, they need to use them effectively to create mental representations of these complex cognitive processes (Gilmore, 1991 & Lin et al., 2016). All these complexities during programming have a possible experience of frustration and difficulty when novice programmers do not acquire debugging skills at an acceptable level (Perkins & Martin, 1986).

Due to its critical importance in programming, various studies have examined the factors that affect the debugging skills of novice programmers (Putnam, Sleeman, Baxter, & Kuspa, 1986; Murphy et al., 2008). Besides, researchers who wish to conduct a systematic review for debugging processes seem to focus on the cognitive processes of students during debugging (Brooks, 1977; Renumol, Janakiram, & Jayapraksh, 2010; Xu & Rajlich, 2004). The common point of all these studies is to examine the pedagogical tools and methods that can make the programming skills of the novice programmers more effective. One of these studies emphasized the importance of the teacher's accurate guidance in acquiring debugging skills and offered students should be encouraged to develop the cognitive skills necessary for debugging skills. Besides, he emphasized that it would be essential to develop adaptive teaching strategies and educational media. However, to develop these strategies and media, it is necessary to study the cognitive processes in debugging processes (Lin et al., 2016).

Debugging in programming instructions pinpointing the novice debugging strategies has not been a hot research topic in Turkey context. This gap makes it challenging to design instructions for programming in align with the cultural considerations. When we examine learners' question-solving behaviors and cognitive levels on this level, it will be easier for us to identify instructional strategies and design and develop instructional materials for effective programming teaching. This study was framed by Lin et al. (2016)'s suggestions on debugging instruction under two concerns: (i) classification of learners according to programming skills and (ii) the need for paper-based debugging questions to see the relationship between types of programming questions and completion rate of these questions. By framing the study around this gap, problem-solving behaviors and learners' debugging performances characterized by the rates of error detection, error correction, and error completion will help us in offering instructional design issues for potential pedagogical tools considering novice programming learners at different skill levels.

Studies in Programming Instruction

Programming instruction is still a challenge due to its complex nature, which may cause misconceptions on learners' mental models and limited knowledge of what kind of environments may develop learners' programming skills. Cakiroglu (2012) conducted a quasi-experimental design and compared novice programmers' performance between blended and face-to-face sessions to examine the effectiveness of programming instructions in both mixed and face-to-face environments. He mentioned delayed tests showed face-to-face meetings were more effective on the increase in the performance. This finding brought us to the identification of the factors in instructional design on a computerized programming instruction environment for enhancing effectiveness. Although studies utilize the high correlation between problem-solving skills and programming, another point of view is having potential misconceptions of crucial concepts during learners engage in programming instructions. To specify these key concepts held by novice programmers, Ma, Ferguson, Roger, and Wood (2011) investigated the mental models of novice programmers and proposed a visualization program to scaffold novices by giving feedback to foster learners' understanding of key programming concepts.

Similarly, supporting novice programmers while they are engaging in programing instructions is a critical factor for improving learners' programming skills. Rum and Ismail (2017) examined metacognitive-related strategies that support programming learning. They proposed metacognitive scaffolding, reflective prompts, self-questioning, self-directed learning, and graphic organizers for computer-assisted programming learning.

Visualization in programming instruction and metacognitive support have been supported by findings as they have effective outcomes in programming comprehension, integrating these factors with multimedia courseware is an essential aspect in novice learners' understanding of abstract concepts in programming

(Annamalai & Salam, 2017). The investigation of tools providing metacognitive support for supplementing traditional courses. Fabic, Mitrovic, and Neshatian (2018) offered a mobile Python tutor platform, including activities for supporting debugging, code-tracing, and code writing skills. Their results revealed advanced programmers differed in terms of code writing skills compared to novice programmers, and novice programmers could not show any learning gain for coding writing. Although they found debugging others' programmers is beneficial for advanced programmers who require higher-order skills compared to code writing, it was not effective for novice programmers.

On the contrary, a longitudinal study has been offered the effectiveness of debugging for novice learners as well (Ahmadzadeh et al., 2005). Similar to this finding, Lee, Yu, Tang, Wong, and Poon (2017) experimented with the effectiveness of a virtual debugging advisor for novice programmers. This interactive environment was designed to analyze the students' programming outputs and suggested potential mistakes for them as an alternative for debugging. The results showed that, along with increased motivation of novice programmers during engaging in virtual debugging advice tool, provided feedback for them to foster their understanding of errors.

Learner Characteristics in Debugging Researches

Learner characteristics, including dichotomies such as expert/novice, experienced/inexperienced, or successful/unsuccessful, have an essential role in research focusing on debugging. Many of the research studies on debugging classified the learners based on these dichotomies and reported learning process and outcomes of these students by adhering to them (Bednarik, 2012; Fitzgerald et al., 2008; Lin et al., 2016; Rodrigo, Andallaza, Castro, Armenta, & Jadud, 2013).

Rodrigo et al. (2013) examined the data log of novice programmers based on their success rates on debugging and they have concluded that low-achieving and high-achieving students differ in understanding the basic programming concepts. In addition, Fitzgerald et al. (2008) classified novice learners based on their high and low success rates and examined the relationship between programming and debugging skills. The results showed that novices who are successful at debugging are also successful in programming activities; however, the novices who are successful in programming activities did not achieve the same success rate in debugging.

In another study, novice programmers classified based on their success rates, as being experienced or inexperienced (Bednarik, 2012). The study examined the visual attention strategies of experienced and inexperienced programmers and revealed that novices often go back and forth between code and output, however, experienced learners showed behaviors on using their prior knowledge while they are associating the code with output in a more systematic way. Similarly, Lin et al. (2016) have achieved similar results. They revealed that while low-performers showed flawless debugging behaviors and focused on syntactical details rather than the logical side of the codes. On the other hand, high-performers tend to break the code into meaningful pieces systematically and develop debugging strategies based on their prior knowledge.

Methodologies in Debugging Researches

Research studies conducted on debugging in programming show methodological differences. These methodological differences are mostly depending on the target group, type, and content of the used materials and research method. Yen, Wu, and Lin (2012) conducted a qualitative study on novice and expert programmers to classify the difficulty levels among different types of errors. They asked participants to think aloud throughout their debugging behaviors while using a compiler for syntax, semantic and logic errors. Study results showed that during the debugging process, syntax errors are more straightforward compared to semantic errors and semantic errors are more manageable compared to logic errors. In the study, two debugging exercises were presented to participants with a total of six errors, each with syntax, a semantic, and a logic error. At the end of the study, it was found that experienced programmers have a better understanding of the feedback received from the compiler, show similar thinking processes with novices during the debugging process, and predict the output of semantic errors. Besides, it is reported that novice programmers experience confusion while they are debugging for semantic errors and make a backward inference. This study reported the debugging time, the completion time and the frequency of debugging the behavior. The frequency of debugging behavior was investigated in seven phases as comprehension, questioning, estimation, execution, observation, correction and confirmation. Also, Lin et al. (2016) assured that think-aloud protocol results in cognitive load during debugging so that to observe students' insight without verbalizing their actions, they preferred to use eye-tracking methodology. They conducted retrospective interviews with each participant after each programming task. Lin et al. (2016) reported that for debugging tasks, a computerized environment with note-taking field and conducting retrospective interviews after each task pose validity thread for an eye-tracking study.

Besides the difficulty levels among types of errors and differences powered by student characteristics, qualitative findings on students' markings during debugging may facilitate in understanding students' debugging behaviors. Ahmadzadeh, Elliman ve Higgins (2007), in their longitudinal experimental research with university students, observed changes in the markings on activities given as students' debugging experiences increased in 15 activities spread over the period. In the study, the number of errors and types of errors that students did during 15 weeks, online programming exam notes, and semantic errors was reported. At the end of this study, although the number of errors in conditional expressions, loops, methods, arrays, classes, and objects decreased for each student, no significant difference was found.

Nivala, Hauser, Mottok, and Gruber (2016) used Gilmore and Green's error classification framework in their work and prepared codes based on a surface level, the flow of control, structure of the plan and structural interaction. There are eight tasks in the study, and each task includes two stages, which are reading and response stages. They determined the reading phase as the time duration until the programmers began to write after they have shown the code, and the response phase includes the time duration, which they started to write until the question ended with pressing the end key. As a result of the study, experienced programmers reported a shorter saccade length and focus duration during the reaction phase and shorter focus durations in the reading phase. They also suggested for future studies that debugging durations for analysis should be added.

Intelligent E-Learning Systems in Programming

Recent researches have been focusing on students' behaviors to enhance their learning outcomes on intelligent e-learning systems within a variety of course domain such as mathematics education (Ozyurt, Ozyurt, Baki & Guven, 2014), and computer programming (Huang, Chen, Luo, Chen & Chuang, 2008). Also, based on students' profiles, systems may refer to personalized learning outcomes and procedures to them to enhance the educational effectiveness in those domains. Knowledge tracking via computer-aided learning or intelligent e-learning platforms seems to have become an essential way in programming education as well for personalized learning (Trifa, Hedhili & Chaari, 2019).

The personalization processes in the e-learning environment require a flexible and adaptive system for each student. Markovic, Kadoic, and Kovacic (2018) investigated the adaptivity criteria for a proposed prototype of an intelligent e-learning system. They mentioned that the selected adaptivity criteria, which are students' cognitive style, learning style, and learning goals, could be included for an adaptive intelligent e-learning system for better predictions of their learning progress. Moreover, Trifa, Hedhili and Chaari (2019) classified each students' response or question via an intelligent agent so that the accuracy rate of the students' learning outcome prediction was measured by considering any interaction that students make throughout their learning process. And, Trifa et al. (2019) discussed adding mod-related influencing factors such as stress, anxiety, and depression to have a better prediction rate of their learning outcomes.

By focusing on programming instruction, Huang, Chen, Luo, Chen, and Chuang (2008) proposed an e-learning tool for introductory programming courses based on an intelligent diagnosis and assessment. Their experimental study suggested that low-performance learners can be assisted effectively by providing immediate feedback after diagnosing learners' source codes and comments. Although this course of work constructs a learning diagnosis model for the learners during their planning of wrong solutions to guide them, influencing factors such as gender, grading, and other collaborative mechanisms are absent in the diagnosis model.

Similarly, a study emphasizing the importance of personalization in intelligent tutoring systems described student modeling as a critical factor (Chrysafiadi, & Virvou, 2013). The student-model keeps information about students' current and prior knowledge, misconceptions, knowledge gap, interests, etc. (Durrani & Durrani, 2010). Chrysafiadi and Virvou (2013) developed a method called PeRSIVA, which evaluates the effectiveness, validity, and accuracy of a student model. This model is a web-based learning environment for teaching C programming language. The study revealed that student performance is affected significantly by the utilized model. Besides, Liang et al. (2017) designed a student profile model by studying students' online learning behaviors at a deep level. They claimed that for the development of effective e-learning environments, the student-profile model has importance.

The studies mentioned above on intelligent e-learning systems show that tracking students' cognitive, metacognitive, and emotional knowledge is a complicated problem. For programming instruction, factors affecting students' programming behaviors are still a question mark to be asked. Mining debugging processes may bring a piece of knowledge about what kind of facilities should be embedded into the intelligent e-learning platforms to predict students' programming performance better and guide them in a way about their programming behaviors. This study constructs a foundation for this future e-learning platform by detecting students' debugging behaviors in a paper-based environment.

METHOD

Research Design

This study aims to provide a comprehensive and in-depth investigation of the debugging process in programming teaching in terms of cognitive and metacognitive aspects, based on programming students who demonstrate low, medium, and high programming skills and propose instructional strategies for scaffolding novice learners in an effective way. The research design that allows a detailed investigation of a subject, event, and person context is the case study. The case study is expressed as an in-depth examination of an environment, subject, person, or a particular event (Merriam, 1988; Yin, 2013; Stake, 1995). Bogdan and Biklen (2007) described the case study with a funnel metaphor. The case study begins with a wide opening, like a funnel, and the gap narrows as it moves downward. In case studies, the researchers aim to discover potential participants and places that may be the subject or source of the study. They try to find a clue on what is feasible and how to continue in the process. They think about when and how to collect data and focus on the point in which they want to move the study. When necessary, they can put the old plans aside and create new ones. In this process, the subject, place, participant, material, and theme of the research activities become more specific. That is, they move from broader and exploratory beginnings to more structured data collection and analysis.

In the present study, the case study was selected as the research methodology. The researchers agreed on the content of the paper-based debugging questions in line with a comprehensive literature review and elaborated discussions. Accordingly, it was decided to include course type (C++, Python) types of errors (syntax & logic), question types (if-else & while), question difficulty levels (low, moderate, & high), marking types (marking availability in general, note, strikethrough, underline, circle, and arrow), and course grades. When data collection started, the researchers have attempted to change the plan, if necessary. Thus, the aim of the research became more specific and progressed in a more specific direction. Since in case studies, generalizing the results is one of the major concerns, in this study, purposeful sampling is used rather than random sampling (Mills, Durepos, & Wiebe, 2010).

Participants

By using a purposeful sampling method, a total of 41 students (*Female*=30, *Male*=11) who were taking either Python or C++ programming language courses for the first time by the same instructor participated in the study. 5 of the 41 students were graduated from a vocational high school. Therefore, 12,2 % of the students (n=5) had a background in programming education and 87,8 (n=36) had not. Table 1. represents student demographics. The data were collected in basic programming courses given at the Department of Computer Education and Instructional Technology in a public university in the 2017-2018 Fall semester. Due to the difficulty levels of the problems (low, moderate, and high), error types (syntax and logic), and question type (if-else and while), a total of 12 different data (3x2x2) were taken from each student (N=492).

Table 1. Frequency of high school graduation of novice learners by gender

	Gender	
School Type	**Female**	**Male**
Vocational	2	3
Not vocational	28	8

Data Collection

During two weeks, students were given paper-based codes with syntax and logic errors in three different difficulty levels, in the subjects of if conditionals and while loops. Each question was chosen with the items used in previous studies and expert programmer approval. Students were given the following warning before they start debugging: "Please refer to the codes presented to you and decide if there are any errors in the codes. If there is an error you found, specify it, explain the cause(s) of the error, and correct it."

Data Analysis

The collected data were analyzed by the Chi-square test via SPSS 24. There are four different types of question characteristics regarding paper-based debugging questions. First one is *course type,* which is a categorical variable having two subcategories as and *C++*. The second one is the *error* type with two subcategories as *syntax* and *logic*. The third one is the question type, with *if-else* and *while* subcategories. The final one is the *question's difficulty level* having three subcategories named as *low*, moderate, and high. Based on these questions and due to students' varying frequent attempts in detection, correction, and completion of errors, students' answers were categorized into three variables as *error detection, error correction attempt,* and completion of error correction. All these variables have two subcategories as *yes* and *no*. In addition to detection, correction, and completion variables, students' markings on the paper-based question were categorized into six types as *marking availability, note-taking, strikethrough, underline, circle, and arrow*. Similar to students' observation of metacognitive outcomes, these categories were also coded as *yes* and *no*.

FINDINGS

Chi-Square Tests for Error Detection, Error Correction Attempt and Completion of Error Correction

A chi-square test was performed to examine the relationship between the question's *difficulty* level and error detection. As demonstrated in Table 2, the relation between these variables was significant (X^2 (2) = 11,05, p = .004, Crammer's V=.15). Novice programmers are more likely to detect errors in low difficulty level of questions than moderate and high difficulty level of questions. Besides, the *question's difficulty* level has a small to moderate effect on error detection.

Table 2. Frequency of novices' error detection by question difficulty level

	Question Difficulty Level		
Error detection	Low	Moderate	High
Detected	97	72	70
Not detected	67	92	94

X^2 (2) = 11.05, p = .004

In order to examine the relationship between the *question's difficulty level* and *error correction attempt,* a chi-square test was performed. The relation between these variables was significant, (X^2 (2) = 12,17, p = .002, Crammer's V=.16) (See Table 3.). Post Hoc analysis revealed that error correction attempt was high in low difficulty level of questions than moderate and high difficulty level of questions. Besides, the level of difficulty has a small to moderate effect on error detection. A chi-square test revealed no significant relationship between the *question's difficulty level* and *completion of error correction*.

Table 3. Frequency of novices' error correction attempt by question difficulty level

	Question Difficulty Level		
Error correction attempt	Low	Moderate	High
Corrected	92	63	67
Not corrected	72	101	97

X^2 (2) = 12.17, p = .002

A chi-square test was performed to examine the relationship between *error type* and *error detection*. As seen in Table 4, the relation between these variables was significant (X^2 (1) = 7.82, p = .005, Crammer's V=.13). Students are more likely to detect *logic errors* than *syntax errors*. Besides, the error type has a small to moderate effect on error detection. A chi-square test revealed no significant relationship between *error type* and *error correction attempt*. Similarly, no relationship was found between error type and completion. Finally, the relation between the question type (*if-else, while*) and the *error detection*, correction attempt, and *completion of error correction* are not significant.

Table 4. Frequency of novices' error detection by error type

	Error Type	
Error detection	Syntax	Logic
Detected	104	135
Not detected	142	111

X^2 (1) = 7.82, p = .005

Chi-Square Tests for Paper-Based Markings

A chi-square test was performed to examine the relationship between *question's difficulty level* and *marking availability* both in general and under the aforementioned categories as a note, strikethrough, underline, circle, and arrow. A significant relation with small to moderate effect was revealed between the questions' difficulty levels and marking availability (X^2 (2) = 8.20, p < .05, Crammer's V=.13) (See Table 5.), note (X^2 (2) = 14.26, p < .01, Crammer's V=.17) and underline (X^2 (2) = 7.08, p < .05, Crammer's V=.12). To examine the relationship between questions' difficulty level and marking availability, Post Hoc analysis was performed. According to Bonferroni correction results, significant relations between question in high difficulty level and marking availability (X^2 (2) = 8.20, p = 0.006), question in high difficulty level and note (X^2 (2) = 14.26, p = 0.002) and question in low difficulty level and note (X^2 (2) = 14.26, p = 0.002) were revealed.

Table 5. Frequency of novices' marking availability by question difficulty level

	Question Difficulty Level		
Marking availability	Low	Moderate	High
Marked	101	94	76
Not marked	63	70	88

X^2 *(2) = 8.20, p < .05*

A chi-square test was performed to examine the relationship between *question's error types* and *marking availability* both in general and under the categories as mentioned earlier as a note, strikethrough, underline, circle, and arrow. A significant relation with small to moderate effect was revealed between the questions' error types and marking availability (X^2 (1) = 12.49, *p* < .01, Crammer's V=.16) (See Table 6.), note (X^2 (1) = 27.53, *p* < .01, Crammer's V=.24) and strikethrough (X^2 (1) = 12.18, *p* < .01, Crammer's V=.16). To examine the relationship between question's error types and marking availability, Post Hoc analysis was performed. According to results, Bonferroni correction did not reveal any significance between variables.

Table 6. Frequency of novices' marking availability by error type

	Error Type	
Marking availability	Syntax	Logic
Marked	116	155
Not marked	130	91

X^2 *(1) = 12.49, p < .01*

A chi-square test was performed to examine the relationship between the *question's types* and *marking availability* both in general and under the categories as mentioned earlier as note, strikethrough, underline, circle, and arrow. Chi-square results did not reveal a significant relationship. Moreover, no significant relationship was revealed for marking availability in the comparison between Python and C++ courses.

DISCUSSIONS AND CONCLUSION

The aim of this study was to investigate the relationship between debugging outcomes of novice programmers and various question characteristics and propose instructional strategies for scaffolding novice learners in an effective way. Results revealed that error detection and correction are high in low difficulty level questions. The level of difficulty was found to have a low-to-moderate significant negative relationship with both error detection and correction. The reason for these meaningful negative relationships (in favor of low difficulty level questions) may be due to that programming requires keeping various information at different levels in the working memory. This information is stated as syntactic details, mental models for problem-solving and specifying the solution for problems (Balzert, 2004; Rist, 1995; Stachel et al., 2013). As the level of difficulty increases, the load on the working memory also increases, considering the requirements of programming. Therefore, the decrease in the problem-solving rates of moderate and high difficulty level questions may be related to this increase. There are several research studies conducted on instructional strategies like scaffolding and decreasing the cognitive load. Sweller (2008) claimed that instructional strategies and activities, which include scaffolding systems, have a big influence on cognitive load. Directly linked to the current study, Stachel et al. (2013) provided scaffolding tools to the programming learners in their experimental research, and results showed that the tool decreased their cognitive load and made a modest contribution to course scores. Similarly, Salleh et al. (2018) generated a scaffolding tool based on cognitive load theory and decreased cognitive load of novice learners. Based on the literature, the difficulty level of the question can be one of the factors in the development of the scaffolding tool.

There was a significant low to moderate relationship between error type and error detection and a low and significant relationship between error type and error correction attempt. Novice programmers detected logic errors more than syntax errors. These results are surprising, given that debugging logic errors require higher cognitive activities than debugging syntax errors. However, several studies claim that it is more complicated than it seems to handle syntax errors (Denny et al., 2014; Denny, Luxton-Reilly, Tempero, & Hendrickx, 2011; Fitzgerald et al., 2008; Nienaltowski, Pedroni, & Meyer, 2008). For novice learners, detecting syntax errors, understanding compiler messages and doing right corrections are more difficult compared to thinking on that error.

Moreover, logic errors mainly addressed in programming courses and syntax errors are perceived as superficial errors to address (Denny et al., 2014; Denny et al., 2011). It can also be a factor that affects the results of this study. Besides, since syntax errors are detected by the compiler, and the user is provided with error codes for correcting, novice programmers may not need to keep such errors in the working memory and therefore, those errors may have less detected; as a result, they are less attempted to be corrected.

Significant interaction was found between academic achievement and error detection, error correction attempt and completion of error correction supporting the previous studies in debugging (Bednarik, 2012; Fitzgerald et al., 2008; Lin et al., 2016; Rodrigo, Andallaza, Castro, Armenta, & Jadud, 2013) Similar to other studies' findings, student characteristics (expert/novice, experienced/inexperienced, successful/unsuccessful etc.) have an essential role in debugging. These studies classified the learners according to their skills in programming and reported the study process and its results by adhering to this distinction (Bednarik, 2012; Fitzgerald et al., 2008; Lin et al., 2016; Rodrigo, Andallaza, Castro, Armenta, & Jadud, 2013). One of the studies examined the data logs of novice programmers at the time of compilation and concluded that low and highly successful students differ in understanding the basic programming concepts (Rodrigo et al., 2013). Another study showed that participants who have achieved similar results are trying to sort out the error flawlessly and remain in the details of the syntax, while high-performance participants tend to systematically code the code into pieces which means that they tend to develop debugging strategies according to their current knowledge (Lin et al., 2016). Although our finding on a significant relationship does not reveal a qualitative difference in terms of learners' characteristics, it is a crucial factor to find significance for each learners' problem-solving behaviors and how this is related to their achievement levels. The relationship between these factors may show us the more students practice programming in a detailed manner, the more they gain programming learning. As learners practice programming questions by taking feedbacks and being supported in a metacognitive way, it is easier to develop insight regarding key programming concepts and skills (Rum & Ismail, 2017; Ma, Ferguson, Roper & Mood, 2011).

Moreover, when novice programmers' marking behaviors on paper were examined, a significant relationship was found between the difficulty level of the questions and the marking presence on paper and between the error types and marking presence on paper. In the significant relationship between the type of error and the frequency of taking notes, it was seen that the students took more notes for logic errors compared to the syntax errors, and the frequency of taking notes decreased as the difficulty level of the questions increased. When the marking behaviors on paper are examined, it can be attributed to three reasons that students have more markings on paper during debugging logic errors: (i) reducing the cognitive load, (ii) guiding the problem solving with the aim of guiding itself, and (iii) increasing the error awareness by taking visual feedback. By considering these reasons, it is thought that visual signs, which can be prepared for the learners, can form the feedback leg of the debugging training and thus reduce the cognitive load of the learners during the use of mental resources. To this end, during the debugging questions embedded in the paper-based or computer screen, considering Mayer's (2014) cognitive load-reducing principles within the cognitive theory of multi-learning can play a facilitating role for a novice learner to recognize the set of syntax and logic errors. Similarly, Stachel et al. (2013) examined the effect of scaffolding tools for decreasing the cognitive load of learners for the Visual Basic programming assignments and revealed that they are effective for decreasing cognitive load. Although the cognitive load is a fuzzy field of the research area to be integrated into debugging in programming researches, the reason why learners could not demonstrate more note-taking behaviors in high difficulty level questions may show their needs to be supported. Similar to this finding, Fabic et al. (2018) found that debugging could not reveal high learning gains for novice programmers and even 39% of advanced programmers were reported as competent in debugging.

Markings on paper-based debugging questions revealed that as the difficulty level of the problems increased, the frequency of taking notes of the learners decreased. The reasons for this can be discussed by considering the difficulty level and types of questions. Syntax errors require preliminary information and brief interventions; whereas, logic errors involve processes of intuitive problem solving and require understanding the result of the code. While the difficulty levels of questions may be raised by short interventions, students may need to refer to their mental resources more during problem-solving as the difficulty level of the questions increases. With the rise in the number of lines in the code, students' negative attitudes towards the question may be blocking their taking notes for logic errors in the questions having long blocks. To overcome this gap, debugging training may be planned to be prepared exhibiting a gradual teaching approach in the questions containing long code blocks can provide the students to develop a positive attitude. In addition, note-taking behaviors for logic errors were found to vary. The reason for this finding can be seen as students' use of intuitive problem-solving behavior and trial/error strategy, and supporting working memories to reach a solution, but also to keep multiple information and thus reduce cognitive load. With this finding, the strategies supported by the students can be supported by the computer-aided debugging programs having the opportunity to see the students' previous markings can be prepared so that students can get tips based on their current behaviors or selected strategies.

Finally, scaffolding tools for novice learners may be effective in decreasing cognitive load and increase the effectiveness of the programming course. However, in programming courses, there are students at different expertise levels regarding their programming abilities. Kalyuga (2009) claims that although strategies developed for novice learners based on cognitive load theory may improve understanding, learners with high expertise can be influenced by these strategies in a negative way. Moreover, prior knowledge is seen as one of the most influential learner characteristics that decide the success of instructional tools on learning (Kalyuga, Chandler, & Sweller, 1998). Therefore, learners need to be supported regarding their characteristics by the scaffolding tools. Mayer (2014) suggests the personalization principle allows learners to interact with computers in a conversational style to improve motivation and learning. Therefore, the scaffolding tool to be developed should use a language that aims to keep interaction with the learner at a high level and encourage the learner to communicate as far away from formality as possible.

This study focused on the aspects of how novice programming students performed during debugging processes on the paper-based debugging question materials to construct a foundation for future intelligent e-learning platforms. The relationship between the result of error-detection, error-correction, and error-completion and the marking availability was stated as significant. These findings are essential to offer student-model for intelligent e-learning environments during the prediction of their performance. Previous researches suggested that cognitive, metacognitive, and emotional factors influencing students' performance are crucial for better prediction (Huang, Chen, Luo, Chen & Chuang, 2008; Trifa, Hedhili & Chaari, 2019). Thus, visual guidance can be provided in a personalized way within these environments.

RECOMMENDATIONS

The results of the study have recommended potential effective strategies for debugging syntax and logic errors regarding stepwise instruction, guided hints, and visualization issues. First of all, there may be stepwise instruction for novice programmers to make significant contributions for debugging in high-level difficulty questions. Secondly, since syntax errors may not be detected by novice programmers due to instant return of errors in compilers, novice programmers may be trained for writing code blocks on paper-based environments by referencing to their stepwise instruction, and they may be guided by code blocks for guiding students with hints in a computerized environment for detection of syntax errors. As a final recommendation, since looking at students' debugging behaviors of solved paper-based programming questions provided in-direct observation for their debugging processes on how, when and in which question types they demonstrate marking behaviors. Based on these observations, supporting working memory by having a note region on both paper-based and computerized environments may be recommended. In addition, the results may suggest that novice programming students may need visualized guidance, and in debugging training, students may be provided such kind of visualized guidance to students for self-guidance. Although the presence of different types of markings did not reveal significant relationships in this study apart from note-taking behaviors, the follow-up analysis should be conducted for prediction of the outcomes as error detection, error correction attempt and error completion by each marking type and types of questions. This may bring personalized visual guidance for each type of question and error types for novice learners within e-learning environments.

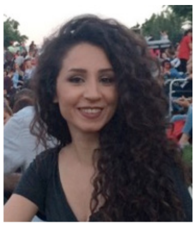
LIMITATIONS

Limitations of this study are three-folded as (i) study duration, (ii) study environment, and (iii) unfamiliarity of the exam questions. First of all, this study endured two weeks, so that limited the observation of programming topics with if-else conditionals and while loops. Secondly, learners took paper-based debugging questions individually in a laboratory setting as a whole class. Therefore, it was not possible to observe and record learners' think-aloud data. Finally, exam questions did not include any types of debugging questions used in paper-based study protocol so that students could not be followed in regard to their debugging behaviors.

BIODATA and CONTACT ADDRESSES of AUTHORS

Gamze TURKMEN is a research assistant and PhD student at Computer Education and Instructional Technology Department, Middle East Technical University. Gamze gained her M.Sc. in Cognitive Science in September, 2013. Her academic interest areas are instructional design, human-computer interaction, metacognition, eye-tracking methodology in learning sciences, and neuroscience of learning. She has been studying the metacognitive processes of secondary school students during collaborative tasks within science center environments to inform the instructional designers, policymakers, science center educators and managers, and also teachers. Her primary qualitative research data includes synchronous eye-tracking video-data of two peers.

Gamze TURKMEN
Computer Education and Instructional Technology, Faculty of Education
Address: Middle East Technical University, 06800, Ankara, Turkey
Phone: +90 3122107523
E-mail: gturkmen@metu.edu.tr, gamze.trkmn@gmail.com



Sonay CANER is a Research Assistant of Computer Education and Instructional Technology at Faculty of Education, Middle East Technical University. She is a Ph.D. candidate in Computer Education and Instructional Technology. Her academic interest areas are Internet use behavior, problematic Internet use, programming instruction, self-regulation, self-regulated learning, educational interfaces, e-learning, use of the internet and new media in education. In addition, she studies the development of mobile intervention applications targeting decreasing problematic Internet use in educational contexts. She has one journal article published in SSCI, a number of papers submitted to international meetings.

Sonay CANER
Computer Education and Instructional Technology, Faculty of Education,
Address: Middle East Technical University, 06800, Ankara, TURKEY,
Phone: +90 312 210 7523
E-mail: csonay@metu.edu.tr, sonaycaner@gmail.com

REFERENCES

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005). An analysis of patterns of debugging among novice computer science students. Acm sigcse bulletin, 37(3), 84-88.

Annamalai, S. & Salam, S. N. A. (2017). Facilitating Programming Comprehension for Novice Learners with Multimedia Approach: A Preliminary Investigation. In *Proceedings of AIP Conference 1891,* 020029.

Bednarik, R. (2012). Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations. *International Journal of Human Computer Studies*, *70*(2), 143–155. https://doi.org/10.1016/j.ijhcs.2011.09.003

Brooks, R. (1977). Towards a theory of the cognitive processes in computer programming. International Journal of Man-Machine Studies, 9(6), 737-751.

Cakiroglu, U. (2012). Comparison of novice programmers' performances: Blended versus face-to-face. *Turkish Online Journal of Distance Education.* 13(3), 135-151.

Chrysafiadi, K., & Virvou, M. (2013). PeRSIVA: An empirical evaluation method of a student model of an intelligent e-learning environment for computer programming. *Computers & Education*, *68*, 322-333.

Denny, P., Luxton-Reilly, A., & Carpenter, D. (2014, June). Enhancing syntax error messages appears ineffectual. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 273-278). ACM.

Denny, P., Luxton-Reilly, A., Tempero, E., & Hendrickx, J. (2011, June). Understanding the syntax barrier for novices. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 208-212). ACM.

Durrani, S., & Durrani, D. S. (2010). Intelligent tutoring systems and cognitive abilities. In *Proceedings of graduate colloquium on computer sciences (GCCS).*

Fabic, G.V.F., Mitrovic, A. & Neshatian, K. (2018). Investigating the effects of learning activities in a mobile Python tutor for targeting multiple coding skills. *Research and Practice in Technology Enhanced Learning*. 13(23).

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, *18*(2), 93-116.

Gilmore, D. J. (1991). Models of debugging. *Acta psychologica*, *78*(1), 151-172.

Huang, C.-J., Chen, C.-H., Luo, Y.-C., Chen, H.-X., & Chuang, Y.-T. (2008). Developing an Intelligent Diagnosis and Assessment E-learning Tool for Introductory Programming. Educational Technology & Society, 11 (4), 139–157.

Kalyuga, S., Chandler, P., & Sweller, J. (1998). Levels of expertise and instructional design. Human Factors, 40, 1–17.

Ko, A. J., & Myers, B. A. (2003, October). Development and evaluation of a model of programming errors. In Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on (pp. 7-14). IEEE.

Lee, V.C.S., Yu, Y.T., Tang, C.M., Wong, T.L. & Poon, C.K. (2018). ViDA: A virtual debugging advisor for supporting learning in computer programming courses. *Journal of Computer Assisted Learning*, Vol. 34, 243-258.

Liang, K., Zhang, Y., He, Y., Zhou, Y., Tan, W., & Li, X. (2017). Online behavior analysis-based student profile for intelligent E-learning. *Journal of Electrical and Computer Engineering, 2017.*

Lin, Y. T., Wu, C. C., Hou, T. Y., Lin, Y. C., Yang, F. Y., & Chang, C. H. (2016). Tracking Students' Cognitive Processes During Program Debugging—An Eye-Movement Approach. *IEEE Transactions on Education*, *59*(3), 175-186.

Markovic, M. G., Kadoic, N., & Kovacic, B. (2018). Selection and Prioritization of Adaptivity Criteria in Intelligent and Adaptive Hypermedia e-Learning Systems. TEM Journal, 7(1), 137–146. https://doi.org/10.18421/TEM71-16

Mayer, R. E. (2014). Principles of multimedia learning based on social cues: Personalization, voice, and image principles. In R. E. Mayer (Ed.), *The Cambridge Handbook of Multimedia Learning* (pp. 201–212). New York: Cambridge University Press.

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: a review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67-92.

Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., & Zander, C. (2008, March). Debugging: the good, the bad, and the quirky--a qualitative analysis of novices' strategies. In ACM SIGCSE Bulletin (Vol. 40, No. 1, pp. 163-167). ACM.

Nienaltowski, M. H., Pedroni, M., & Meyer, B. (2008). Compiler error messages: What can help novices?. *ACM SIGCSE Bulletin*, *40*(1), 168-172.

Nivala, M., Hauser, F., Mottok, J., & Gruber, H. (2016, April). Developing visual expertise in software engineering: An eye-tracking study. In Global Engineering Education Conference (EDUCON), 2016 IEEE (pp. 613-620). IEEE.

Ozyurt, O., Ozyurt, H., Baki, A., & Guven, B. (2014). Bir Bireysellestirilmis Uyarlanabilir ve Zeki E-Ogrenme Ortami ile Gerceklestirilen Matematik Ogretiminden Yansimalar. (Turkish). Education & Science / Egitim ve Bilim, 39(174), 129–142. https://doi.org/10.15390/EB.2014.1791

Perkins, D., & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In E. Soloway & S. Iyengar (Eds.), Empirical studies of programmers (pp. 213–229). Norwood, NJ: Ablex.

Putnam, R. T., Sleeman, D., Baxter, J. A., & Kuspa, L. K. (1986). A summary of misconceptions of high school Basic programmers. Journal of Educational Computing Research, 2(4), 459-472.

Renumol, V. G., Janakiram, D., & Jayaprakash, S. (2010). Identification of cognitive processes of effective and ineffective students during computer programming. ACM Transactions on Computing Education (TOCE), 10(3), 10.

Rodrigo, M. M. T., Andallaza, T. C. S., Castro, F. E. V. G., Armenta, M. L. V, Dy, T. T., & Jadud, M. C. (2013). An Analysis of JAVA Programming Behaviors, Affect, Perceptions, and Syntax Errors among Low-Achieving, Average, and High-Achieving Novice Programmers. *Journal of Educational Computing Research*, *49*(3), 293–325. https://doi.org/10.2190/EC.49.3.b

Rum, S. N. M. & Ismail, M. A. (2017). Metacognitive Support Accelerates Computer-Assisted Learning for Novice Programmers. *Educational Technology & Society, 20*(3), 170-181.

Salleh, S. M., Shukur, Z., & Judi, H. M. (2018). Scaffolding model for efficient programming learning based on cognitive load theory. *Int. J. Pure Appl. Math*, *118*(7), 77-83.

Savitch, W. J., Mock, K., Bhattacharjee, A. K., & Mukherjee, S. (2012). Problem-solving with C++. Pearson Addison Wesley.

Stachel, J., Marghitu, D., Brahim, T. B., Sims, R., Reynolds, L., & Czelusniak, V. (2013). Managing cognitive load in introductory programming courses: A cognitive aware scaffolding tool. *Journal of Integrated Design and Process Science*, *17*(1), 37-54.

A., Hedhili, A. & Chaari, W.L. *Knowledge tracing with an intelligent agent, in an e-learning platform*. Educ Inf Technol 24, 711–741 (2019) doi:10.1007/s10639-018-9792-5.

Xu, S., & Rajlich, V. (2004, August). Cognitive process during program debugging. In Cognitive Informatics, 2004. Proceedings of the Third IEEE International Conference on (pp. 176-182). IEEE.

Yen, C. Z., Wu, P. H., & Lin, C. F. (2012). Analysis of Experts' and Novices' Thinking Process in Program Debugging. *Engaging Learners Through Emerging Technologies*, 122-134.