Exploring debugging processes and regulation strategies during collaborative coding tasks among elementary and secondary students

Exploring Debugging Processes and Regulation Strategies during Collaborative Coding Tasks among Elementary and Secondary Students

Meghan M. Parkinson, Seppe Hermans, David Gijbels, Daniel L. Dinsmore

Teaching, Learning and Curriculum, University of North Florida, Jacksonville, USA

Training and Education Sciences, Edubron research group, University of Antwerp, Antwerp, Belgium

Correspondence concerning this article should be addressed to Meghan M. Parkinson, Department of Teaching, Learning, and Curriculum, University of North Florida, 1 UNF Drive, Jacksonville, FL 32207, United States. Email: m.parkinson@unf.edu

Exploring Debugging Processes and Regulation Strategies during Collaborative Coding Tasks among Elementary and Secondary Students

Background and Context: Debugging in education increasingly takes place in a collaborative setting. More data are needed about how young learners identify and fix errors while programming in pairs.

Objective: The current study aims to identify discernible patterns in the intersection between debugging processes that children engage in during coding activities and the type of regulation used during those debugging processes (i.e., SRL, CRL, or SSRL); to gain more insight into how these processes can contribute to coding success and thus drive further theory and model development.

Method: Two experiments were conducted in sequential order. The first focused on second-grade students (N= 12) who were asked to program a Code-a-pillar using physical programming blocks. Two coding schemes were used to identify both debugging processes and types of regulation used. The second study used a similar approach but focused on eighth-grade students (N= 30) who were asked to program a Tobbie2 robot using Microsoft MakeCode.

Finding: Results confirm that the integrated and sequential use of all four debugging processes is related to successful coding. Furthermore, we see similar patterns in the overlap between debugging processes and regulation types for both age categories.

Implications: The study highlights specific areas where strategy training could effectively improve collaborative programming efforts. Moreover, our finding challenges the notion that programming and debugging can be reduced to a set of clearly defined consecutive steps. Instead, it highlights the dynamic nature of coding and debugging processes and the importance of including metacognitive and regulatory elements in debugging models.

Keywords: computational thinking; debugging; regulated learning; social learning

Introduction

As our education systems aim to prepare young people to be able to participate in this fast-evolving world, it is no longer sufficient to just teach the use of existing technology but is also necessary for students to understand underlying principles such as programming and computer logic. In 1980 Papert introduced the term computational thinking in his work, *Mindstorms: Children, Computers, and Powerful Ideas*. He argued that computational thinking could be used to create new knowledge and that computing has the potential to improve children's thought processes. Wing (2006) later re-launched this concept with her call to establish computational thinking as a fundamental skill that everyone should possess. This statement made many consider computational thinking an essential skill in the 21st century (Angeli et al., 2016).

In recent years, researchers have focused on issues related to teaching and learning skills, concepts, and practices relevant to computational thinking (Angeli et al., 2016; Bull et al., 2020; Grover & Pea, 2013). This follows from decades of research that examines how cognitive (i.e., skills and strategies used to make progress on a task) and metacognitive (i.e., skills and strategies used to monitor and control cognitive processes) processes influence learning across a wide variety of different contexts (e.g., Dinsmore, 2017). These processes have been the focus of major learning theories that examine individual learning, such as information processing theory and self-regulatory models (e.g., Winne, 1995) as well as theories that focus on learning in social settings and the related models of socially-shared regulation (e.g., Panadero & Järvelä (2015).

At issue is that whole curricula have been developed and implemented without fully understanding how children new to programming approach computational thinking nor which types of individual and group regulation support their efforts to complete programming tasks (Kallia & Cutts, 2022). For example, some curricula present

programming as a collaborative endeavour (e.g., CSinSF), while others provide individually-oriented problem-solving lessons (e.g., code.org). Furthermore, there are core components of computational thinking that seem to be emphasized, however debugging remains a less understood component of computational thinking in education (Liu et al., 2017). Debugging involves finding and fixing errors or "bugs" in computer programs, making it a type of problem-solving strategy (Liu et al., 2017; Klahr & Carver; 1988). Strategies have been defined as a special type of procedural knowledge that individuals need to complete tasks (e.g., Alexander & Judy, 1988). For example, Lin et al. (2016) established a strong correlation between debugging and cognitive activities as previous research has shown that this skill has the potential to be used outside a programming context (Klahr et al., 1988). Although traditional research on debugging posits it as an individual activity (Grover et al., 2013), debugging often happens in collaborative settings, particularly in educational contexts utilizing research-backed curricula (e.g., CSinSF). Therefore, to better understand how students, learn to identify and fix errors while programming in pairs or groups, more data about how learners debug in small groups are needed at various developmental levels. The exploratory data presented in this paper are meant to be a first step to inform explicit instruction for teaching novice programmers how to debug with one or more partners.

Theoretical Framework

Regulated learning in collaborative learning environments forms an emerging and growing area of research, but there is very little knowledge on how regulated learning takes place during collaborative programming and debugging.

Debugging

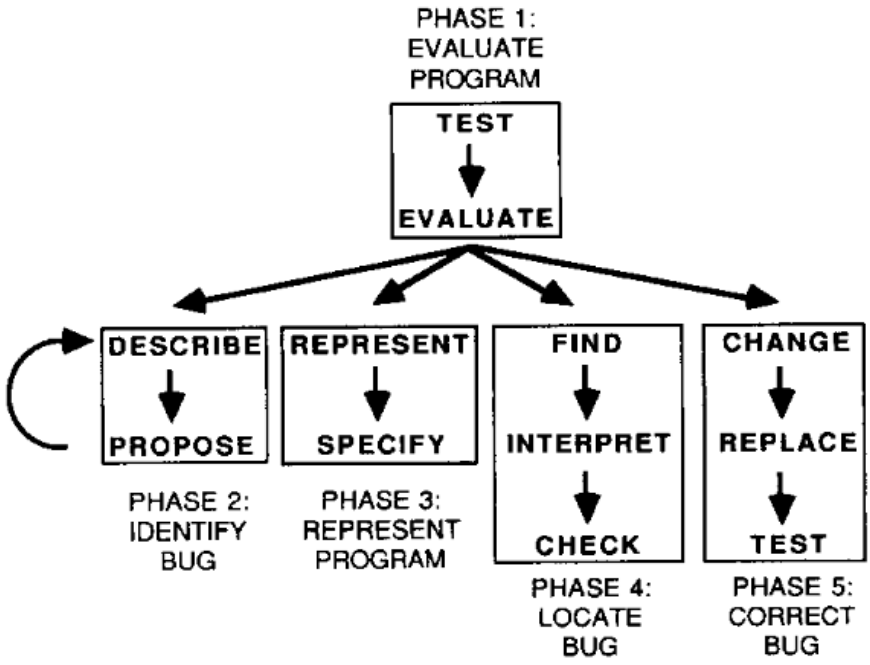
Debugging is defined here as a systematic process to find out why a computer program does not work and attempt to resolve the problem (Liu et al., 2017). The error in the program is also known as a ‘bug’. Debugging forms an established and rich field of study that received much attention in the 1980s. Reports included studies on the differences between inexperienced and experienced programmers, types of bugs, causes of bugs, and mental models of both programming and debugging (McCauley et al., 2008). Klahr and Carver (1988) formulated four aspects that are necessary for debugging: 1) a description of the purpose of the program; 2) a list of steps or commands; 3) the output generated by the program; and 4) individual knowledge of coding. Without the first three aspects, debugging is not possible. The fourth, 'individual knowledge of coding', is more variable and affects debugging success. Novice programmers often lack domain knowledge and general problem-solving skills (Klahr & Carver, 1988; Liu et al., 2017), making debugging particularly difficult for them (Fitzgerald et al., 2008).

More recent studies have explored characteristics of debugging instruction that most effectively improve children’s programming outcomes. Explicitly teaching a debugging process was found to improve both actual debugging and self-efficacy for debugging in high school students (Michaeli & Romeike, 2019). For elementary children, there is evidence to suggest that connecting instruction to familiar and practical language and modeling through concrete means like manipulatives helps them learn debugging (Ahn, et al., 2022; Sung, et al., 2022). The current study seeks to learn more about prior knowledge children bring to a programming task requiring debugging and how they regulate their learning while collaborating on a programming task.

The connection between problem solving and metacognition is frequently cited, partly because debugging can also be considered a problem-solving skill (Klahr &

Carver, 1988). The five stages of debugging (i.e., evaluate, identify, represent, locate, and correct; See Figure 1), proposed by Klahr and Carver (1988), are very similar to stages that can also be found in other problem-solving models (Bruning et al., 2010). Ahn et al. (2017) indicated that these processes were efforts to monitor and control the coding process, similar to how individuals monitor and control their own processes. A corollary with regulatory processes can be made, in which self-regulated learning forms a rich area of study, which we turn to next.

Figure 1. Goal structure of the debugging model. (From Klahr & Carver, 1988. Reprinted with permission.)



Regulated learning

In most theoretical models related to learning, the learning process contains a regulatory mechanism to allow for the adjustment of behavior and thought processes. The ability to control one's own behavior and thought processes is called self-regulation. It is considered critical for successful learning and has been described by Zimmerman (2000) as giving guidance to the learning process to achieve objectives.

Self-regulated learning (SRL) forms a well-established and rich field of study (Panadero, 2017) that focuses on the combination of metacognitive, motivational, and behavioral processes. The metacognitive component includes planning, goal setting, organizing, monitoring, and self-evaluation; the cognitive component includes learning strategies that are used; and the motivational aspect includes motivation, self-efficacy, and interest in the task (Potters, 2014). Regarding collaborative learning, SRL theories have been used to explain regulated learning in more social and interactive learning environments (Järvelä & Hadwin, 2013). These insights drive a relatively new but growing field of research. In what follows, we build on these studies to conceptualize regulation types and processes in a collaborative CT context.

Types of regulation

Multiple frameworks conceptualize the way regulation takes place in a collaborative setting, in which types of regulation vary in function from the "I-perspective ", "you-perspective", and "we-perspective". Järvelä and Hadwin (2013) thus defined three forms of regulation: self-regulation, co-regulation, and shared regulation (see Table 1).

Self-regulated learning (SRL) is essential to be able to work together productively. An individual must first and foremost regulate their own learning (Hadwin et al., 2011). Unlike self-regulation, co-regulation of learning (CRL) is not an individual undertaking. CRL involves guiding and supporting the regulatory processes of one specific group member by another group member (Järvelä & Hadwin, 2013).

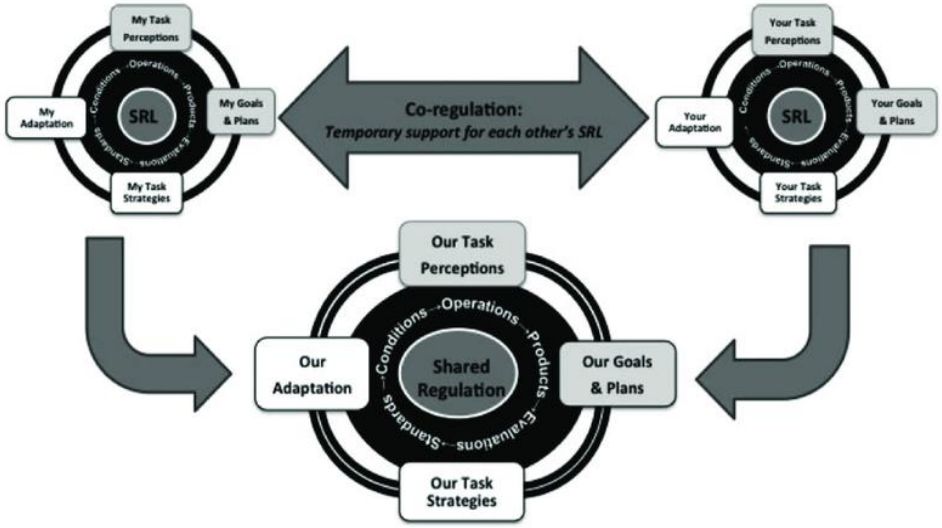
Table 1. Contrasting different regulatory areas in SRL, CRL, and SSRL (adapted from Järvelä, & Hadwin, 2013).

	Self-Regulated Learning (SRL)	***Co-Regulated Learning (CRL)***	***Shared Regulation of Learning (SSRL)***
Whose goals?	("I" perspective) Individuals construct personal goals/standards against which they monitor	("You perspective") Individuals hold goals/standards for	("We perspective") Collective goals/standards are negotiated and constructed amongst team

	their own progress and contributions to the group task.	each other in relation to progress and contributions to the group task.	members to optimize progress and contributions to the group task and to build on goals each individual brings to the group task.
Who regulates?	Individual adapts or changes his/her own regulation processes, beliefs goals, etc.	Individual supports or influences a team member's regulation processes, beliefs goals, etc.	Team members collectively negotiate and realign or adapt group regulation processes, strategies, beliefs, goals, etc.
What is regulated?	My task perceptions. My strategy knowledge & use. My goals and standards for this task. My plans for working together My engagement and positive/negative emotional feelings. My goal progress evaluations.	The other's task perceptions. The other's strategy knowledge & use. The other's goals and standards for this task and for contributing to this task. The other's plans for this task and for contributing to this task. Awareness of others' engagement and positive/negative feelings in this task. Goal progress evaluations of other group members.	Our negotiation of common task perceptions. Knowledge about this group's strengths & weaknesses with respect to this task. Shared goals and alignment of individual task perceptions and goals. Our use of team processes and strategies for succeeding with this task. Strategy knowledge we create together. Perceptions & evaluations of our collective progress. Awareness of our engagement and positive/negative feelings in this task. Negotiated evaluations of goal progress.

The co-regulatory learner provides support and feedback. In this form, an individual is not completely free to monitor and control his or her cognition, motivation, and behavior, but these processes are partially influenced by another person. Finally, socially shared regulation of learning (SSRL; Järvelä and Hadwin, 2013) is an activity in which the regulation of group members is coordinated and discussed, thereby influencing each other. Group negotiation, consultation, discussion, and exchange are characteristic of shared regulation. Malmberg et al. (2017) also indicated that SRL, CRL, and SSRL occur simultaneously and reciprocally (see Figure 2).

Figure 2. Three forms of regulated learning in successful collaboration. (From Järvelä, & Hadwin, 2013. Reprinted with permission.)



Regulation processes

Typically, researchers focusing on regulated learning in a collaborative setting have described how shared regulation arises. Similarly, the role of the individual within a group has been described by proposing different forms of regulatory learning processes. Empirical research on how these processes manifest themselves is less extensive. Zimmerman (2000) described three phases (i.e., preparation, implementation, and reflection) that occur cyclically. The execution phase can be further divided into monitoring and control. Malmberg et al. (2017) in their study among university students who carried out a group assignment focused on which patterns can be found in SRL, CRL, and SSRL; in conjunction with task performance. For this, they categorized the cognitive- and task-related interaction segments during their observation and subsequentially derived theoretically and empirically different categories into which the regulated learning process can be divided (i.e., task understanding, planning, goal setting, monitoring and evaluating, strategy use, and task execution). Although monitoring and evaluation are conceptually two different processes, they were merged as they were found to be difficult to distinguish (Malmberg et al., 2017). In their

categories, the researchers included task performance in addition to the five cognitive segments (see Table 2).

Table 2. Categories representing regulation processes and executive processes and their empirical indicators (adapted from Malmberg et al., 2017).

Processes	***Empirical indicators***
Task understanding	Activating previous knowledge of the task and contents; thinking about the purpose of the task; identifying what should be learned; reading and interpreting the task instructions; explaining and discussing what the midterm plan should look like, thinking about why task completion is important.
Planning	Thinking about what documents and resources are needed (e.g., during the lessons or in a lesson plan); thinking about the relevant parts of a lesson plan. The group plans and coordinates the level of collaboration, such as dividing the task or lesson among group members. Creating a work schedule; planning what to do next.
Goal setting	The group sets a goal for the work to be done; the group sets a task-specific goal, for instance, the need to learn important concepts or theories from the course. The group sets a date, for example, a date for accomplishing a subtask that takes the lesson plan into account. The group decides on the main responsibilities of each member in the lesson plan.
Monitoring and evaluating	Monitoring and evaluating progress toward the criteria set for the task; evaluating the schedule set for the collaborative task; summarizing what has been done and what needs to be done. Monitoring understanding and the resources available.
Strategy use	Prompting the use of strategies such as summarizing information, elaborating on presented information, and selecting main points for the midterm plan.
Task execution	The group works on laptops and writes up the midterm plan (e.g., one group member writes while others explain and contribute to what to write). The group progresses with the content in the midterm plan.

Regulation processes during debugging

In our review of the literature, there were numerous articles related to *debugging, collaboration,* and *regulation,* but few investigated the patterns between these constructs. Therefore, we conclude that research on the regulation of processes during debugging and research on debugging in social educational settings is limited.

However, we found a few studies on debugging that referred to regulatory processes. In their research among college students, Lin et al. (2016) used eye-tracking to investigate the cognitive processes during debugging. They found that differences

between high and low-performing programmers could be explained by their planning skills. Well-performing programmers work in a more structured way and are better at breaking up problems. Liu et al. (2017) came to a similar conclusion during their work with primary school children. They focused on problem-solving strategies that students undertake during debugging. This showed that novice programmers are lacking in problem-solving strategies. With regard to collaborative learning, Murphy et al. (2010) examined the conversations of university students during debugging tasks and found that dyads that communicated more and could complement each other's reasoning had a greater success rate. Villamor et al. (2017) used eye-tracking to investigate how students work together during debugging. Their research also focused on the differences between experienced and inexperienced programmers. They concluded that dyads consisting of two programmers with little experience showed more collaboration than when one or two group members had programming experience. Although interesting, none of these studies had a specific focus on regulatory processes *and* took place in a collaborative setting.

Only recently, Emara et al. (2020) presented a study on debugging and regulatory processes where they analyzed conversations and screen recordings collected during a group STEM assignment that also involved programming. The researchers first identified debugging processes and examined which debugging and regulatory processes could be observed. They relied on theoretical frameworks regarding debugging and SRL, but no mention was made of regulation types that have been identified in a social and interactive environment (Järvelä & Hadwin, 2013; Malmberg et al., 2017).

Purpose of the present study

As research on regulation (i.e., SRL, CRL, and SSRL) during debugging in a social educational environment is limited, our study aims to provide a better understanding of the nature of the debugging processes that students engage in when controlling robots through block-based coding and the type of regulated learning that they exhibit during these debugging processes in a social setting. Moreover, to explore the developmental aspect of debugging in a social setting, the current study presents a cross-sectional design consisting of two experiments with different age groups. Experiment 1 was conducted with second grade students in a North American context while Experiment 2 was conducted with eighth-grade students in a Western European context. While the robots and specific challenges varied to be more appropriate for these different grade levels, important characteristics of the experiments were held constant. The two experiments examined students working in pairs and utilized block-based coding to control robots. Both groups of participants had low socioeconomic status backgrounds and limited prior exposure to coding. The purpose of these data is not to be generalizable (i.e., deductive), but to collect evidence in an inductive fashion to engage in theory and model building (e.g., Chalmers, 1982).

Research questions

1. What debugging processes, if any, did students engage in during the coding activity?
2. For the debugging processes students engaged in, what was the type of regulation that students engaged in during those debugging processes (i.e., SRL, CRL, or SSRL)?
3. Were there any discernable patterns in the intersection between the debugging processes and the types of regulation?

Methods

Students in both experiments were divided into pairs and introduced to a robot. Researchers briefly oriented dyads to the robot and explained the challenge they were to complete in the given time. Each dyad working on a pre-designed task was video recorded and transcribed verbatim by the researchers. Qualitative content analysis was used to analyze the data (Neuendorf, 2017). The analysis was conducted using deductive coding (Saldana, 2015) based on pre-existing frameworks for debugging processes (i.e., Ahn et al., 2017) and regulation types (i.e., Malmberg et al., 2017). Therefore, the coding scheme is grounded in theory that is relevant to the research question. The coding process entailed assigning codes to the data based on their alignment with the pre-existing categories in the frameworks. To ensure the rigor and trustworthiness of the analysis, the researchers who conducted the analysis were trained in qualitative research methods. Additionally, colleagues and other experts in the field were consulted to help validate the findings and ensure their validity. Specific sample characteristics and task differences will be described for each experiment in the following sections. Please note that the exemplars provided are open to interpretation, and different individuals may have different opinions on how they fit the definitions. However, they are meant to be representative and not exhaustive. Moreover, the result of the analysis is based on a consistent application of the codes by the researcher. To provide a more comprehensive and nuanced understanding of our findings, detailed and in-depth accounts are presented in the results section using a narrative format, with quotes and examples

Experiment 1

Materials and methods

Participants

Participants were 12 second-grade students from a large city in the southeastern United States. Students were 60% female and primarily of African American and Latinx heritage, reflective of the student body and surrounding community, and reported little to no experience with coding. They all attended a private school with a mission to serve families in the immediate, primarily low-SES community. The project was reviewed by the first and fourth author's Institutional Review Board and approved (#1305087-4).

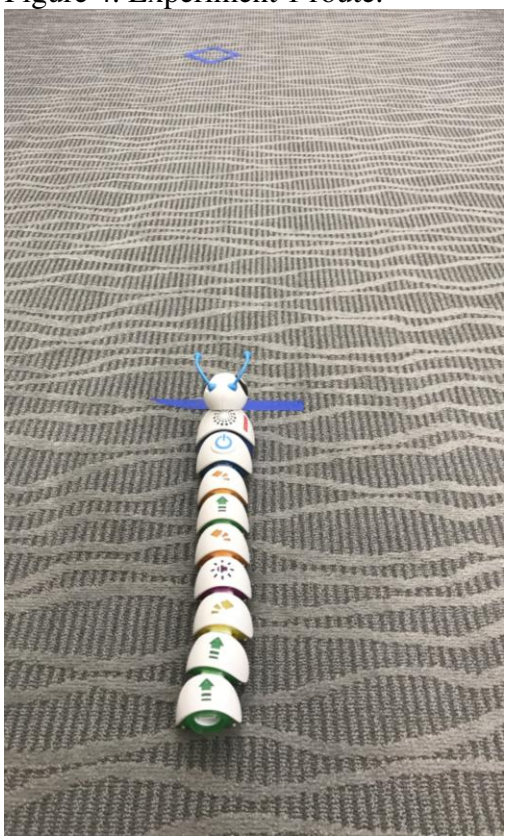
Research task

Participants were asked to use a Code-a-pillar (see Figure 3). The Code-a-pillar is a packaged coding toy that contains a main control unit (the head) and segments that attach to the head and each other via USB attachments. Students code the robot by attaching action pieces that have symbols on them to make the robot either: go straight, turn left, turn right, or play music. Pieces can be attached in any order and the Code-a-pillar will execute each command on the given segment in the order they are attached starting with the head.

Figure 3. Code-a-pillar.

After a brief introduction to how the Code-a-pillar works, students were asked in dyads to program the Code-a-pillar from a start point, go around a block about 8 feet away, then return to the starting point (See Figure 4). Dyads were given 10 minutes to complete this task. Participants were also asked to say anything they were thinking or doing aloud while they performed this research task. These activities were both audio- and video-recorded. All activities took place at the students' elementary school in a large, carpeted room with very little furniture.

Figure 4. Experiment 1 route.



Coding of the audio and video transcripts

All verbal statements were transcribed verbatim from the audio and video recordings with actions (from the video) indicated in the transcripts in brackets. The first and fourth authors coded the transcripts using Malmberg et al.'s previous coding scheme. First, the

fourth author segmented the transcript into codable units. Then, the first and fourth authors independently coded the transcribed segments for episodes of debugging and regulation. An episode was one or several utterances and/or actions by and between students, with the purpose of debugging or regulating. Episodes were considered distinct based on changing foci of utterances or actions before and after the episode or with the passage of time using time stamps. To code for instances of debugging, we developed a scheme that adapted the definition of debugging forwarded by Klahr and colleagues (Ahn et al., 2017; Klahr & Carver, 1988). Table 3 describes the debugging stages, and regulation processes, and gives exemplars of each from the coded transcripts. After this independent coding, these authors compared codes to see if there was consistency. Any disagreement with the codes was rectified through discussion prior to any data analysis.

Table 3. Debugging processes (adapted from Ahn et al., 2017)

Debugging Process	**Related Regulatory Process(es)**	**Conceptual Definition**	**Second Grade Examples**
1. Necessity of debugging	Monitoring and evaluation	Monitoring and evaluating progress toward a goal	"Look, it's not working."
2. Identifying the bug	Task understanding	Activating task knowledge in relation to the discrepancy between the goal and outcome	"I think it's going to go too far past the block that we're supposed to get to."
3. Pinpointing the bug	Planning; Goal setting	Using clues and techniques to figure out how to confirm the bug	"Hang on, I think we need a piece that turns it this way."
4. Correcting the bug	Strategy use; Task execution	Using strategies or enacting processes to confirm that the bug is fixed	"Let's add some more of these pieces and see what happens."

Transcripts were also coded for the type of regulation they engaged in. We adopted the scheme of Malmberg et al. (2017) to identify whether the regulation of the

debugging was either self-regulation, co-regulation, or socially shared regulation. Table 4 presents the three types of regulation, their definitions, and an example episode of each from the coded transcripts. Again, any instances of differences in the coding of the first and fourth authors were rectified in discussion before analysis.

Table 4. Types of Regulation

Type of Regulation	Conceptual Definition	Second Grade Examples	Eighth Grade Examples
Self-regulation	One person is monitoring or controlling the debugging process	"Now it's turning towards me."	"A, I didn't put a stop there, that's why! Okay okay okay."
Co-regulation	One person is prompting others to engage in monitoring or controlling the debugging process, but the other group member is not contributing new information or actions	P1: "Ah, we should take a piece off now." P2: "Yeah."	L1: "What are you going to change?" L2: "Can I not change it?" (L2 adjusts the time) L1: "And now?" L2: "Where does he have to go? A, to the right." (L2 adjusts the program, L1 watches.)
Socially shared regulation	Both group members contribute to the monitoring or control of the debugging process by adding unique inputs to that process	P1: "But we need one more." P2: "But I think it's going to be working this time. I think it might work." … P2: "Wait, you put this piece right here and these three pieces right there." P1: "Put the music to the last one." P2: "Yeah, so we have two musics."	L1: "That one was good." L2: "Are we going to cut that walking down a bit? (L2 adjusts the program) L1: "No, that should be more. Look, he literally has to stop here." (L1 points to the course) L2: "A little more." (L2 points to the track) L1: "If he's going to stop here, he'll go like this." (L1 moves his hands across the course) L2: "Wait a minute." (L2 adjusts the time) L1: "yes"

Results

The coding of the transcripts revealed a total of 260 debugging/regulatory strategies across the 6 dyads of students during coding. These strategies, coded from students' actions and words, included all four aspects of debugging and all three types of regulation (i.e., SRL, CRL, and SSRL). A description of strategies by dyad is presented in Table 5, where episodes refer to strategies coded.

Table 5. Experiment 1: Identified debugging processes and regulation types per dyad.

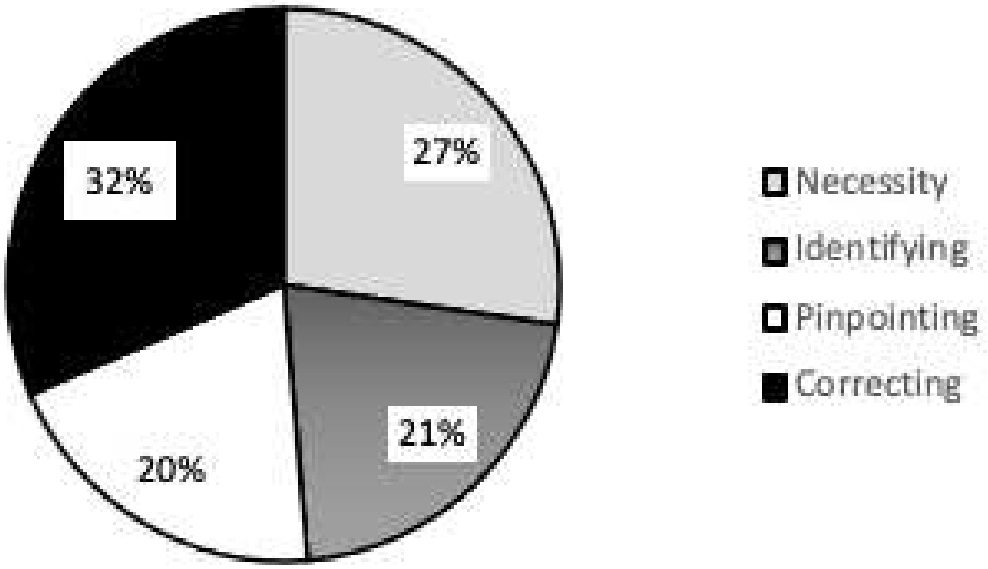
	Number of coded episodes						
	Necessity	Identifying	Pinpointing	Correcting	SRL	CRL	SSRL
Dyad 1	19	11	14	19	7	31	25
Dyad 2	7	7	11	10	7	14	14
Dyad 3	10	9	6	8	6	12	15
Dyad 4	15	11	5	23	1	38	15
Dyad 5	4	7	4	4	7	12	0
Dyad 6	4	1	3	4	1	10	1

Note. SRL= self-regulated learning; CRL= co-regulated learning SSRL= socially shared regulated learning

Debugging processes

For research question one, the quantity of the four processes of debugging was tabulated. Figure 5 displays the percentages of these processes employed across the six dyads. While the differences in quantity were small, the differences in the quality of implementation of those processes were markedly different. Dyad 1 not only employed these processes in higher numbers, but also interleaved these processes and were generally accurate in their attempts to figure out the necessity of debugging, identifying the bug, pinpointing the bug, and correcting the bug. These processes employed by Dyad 1 are contrasted with those of Dyad 6. Rather than interleaving the debugging processes, this group engaged in the same process repeatedly with each other to little avail.

Figure 5. Experiment 1: Use of debugging processes across the six dyads.

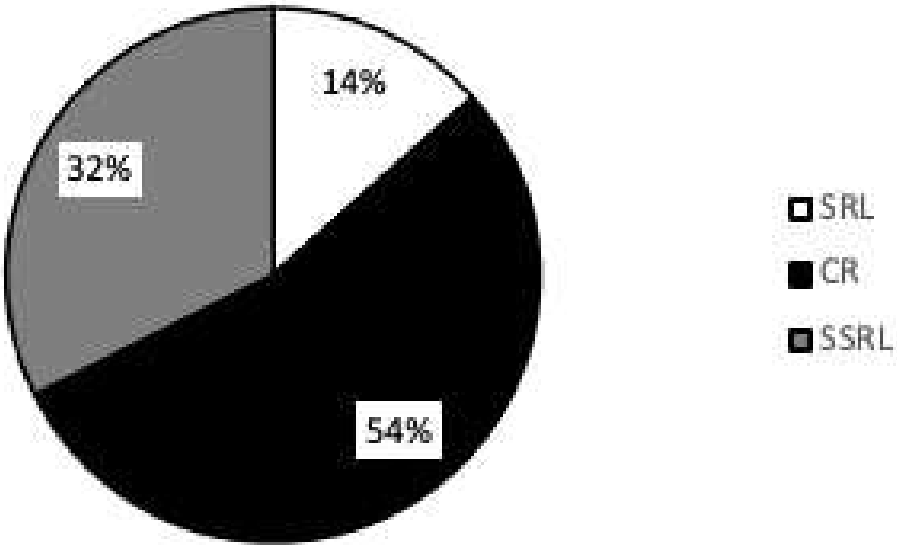


Types of regulation

For research question two, the quantity of the types of regulation was tabulated. Figure 6 displays the percentages of these types employed across the six dyads. Unlike for the processes of debugging, there were significant differences between the types of regulation employed across the groups with co-regulation being employed at much higher rates than SSRL and SRL, respectively. While four out of the six groups followed this general pattern, Dyads 2 and 3 utilized CRL and SSRL types more evenly, with three using SSRL more often than CRL (15 episodes for SSRL versus 12 episodes for CRL). On the other hand, Dyads 5 and 6 barely relied on SSRL at all, with Dyad 6 only engaging in SSRL one time and Dyad 5 not engaging in SSRL at all. Group 1 switched between engaging in SRL and CRL through the first half of the task then pivoted towards interleaving CRL and SSRL through the third quarter of the task and switched entirely to SSRL for the last quarter. They were constantly adding added information and building successfully upon one another's observations, statements, and

directions. This contrasts with Dyad 5, where one of the members took control and the other member of the dyad simply agreed without contributing any added information – an example of CRL. While Dyads 1 and 5 both engaged in patterns of co-regulation, Dyad 5's use of co-regulation was more off-task.

Figure 6. Experiment 1: Use of types of regulation across the six dyads.



Patterns in the debugging and types of regulation

Finally, these data were examined for any evidence certain debugging processes intersected with types of regulation. Table 6 presents the intersection of these data through the percentages, means, and standard deviations of these cross-cutting categories across the six dyads. In addition to CRL being the dominant type of regulation, these data expound upon that by revealing that the focus of CRL is primarily on determining the necessity of debugging (18%) and correcting the bug (19%).

Table 6. Experiment 1: The intersection of debugging processes with types of regulation.

	Percentage	**Mean**	**Standard Deviation**
Necessity SRL	6%	2.00	2.00

Necessity CRL	18%	6.33	4.63
Necessity SSRL	4%	1.50	3.21
Identifying SRL	5%	1.67	1.63
Identifying CRL	10%	3.67	2.50
Identifying SSRL	6%	2.33	1.97
Pinpointing SRL	1%	0.50	0.84
Pinpointing CRL	7%	2.50	1.64
Pinpointing SSRL	12%	4.17	3.49
Correcting SRL	2%	0.67	0.52
Correcting CRL	19%	7.00	6.10
Correcting SSRL	10%	3.67	2.80

Dyads 4 and 6's heavy reliance on necessity debugging and engaging in strategies or action to correct the bug was associated with each dyad member talking over the other in terms of what to do next, with no attempt to connect their ideas to one another or the stated task goal. In contrast, Dyad 2 engaged more often in interleaved debugging processes and interleaved types of regulation. Each successive attempt at cycling through these processes got Dyad 2 closer and closer to getting the Code-a-pillar around the block.

Experiment 2

Results from Experiment 1 imply that models of debugging contain metacognitive and regulatory processes as important patterns in coding. These findings are in line with previous research showing that debugging involves complex cognitive processes (Liu et al., 2017; Fitzgerald et al., 2008). Experiment 1 provides valuable insights, but also has several limitations. For example, it was a small-scale qualitative study with second-grade students. Moreover, programming with physical blocks is rather restrictive. It is therefore opportune to examine how students from an older age group regulate their learning during a collaborative coding assignment. That is why Experiment 2 focuses on how eighth-grade students regulate their learning while programming a robot in small groups. We investigated whether similar connections between debugging and regulation

processes hold for eighth-grade students to gain more insight into how these processes can contribute to coding success and thus drive further theory and model development. One additional research question was added to the same three from Experiment 1:

4. Do any of these patterns of regulations and debugging influence joint coding success?

Materials and methods

Research design

Experiment 2 was designed as an explorative descriptive study in which data were collected through observations. We used the same approach as in Experiment 1, except for the programming task. The programming task was adjusted to be more appropriate for eighth grade students.

Participants

The participants in this study were 30 eighth-grade students (27% female). Students were randomly divided into fifteen dyads. The observations took place at a secondary school in Belgium. The school is characterized by a student population with low SES for which they received funds for this. All students in the study had some experience, albeit limited, with programming and the programming languages (e.g., Microsoft Makecode) used during the assignment. The proposed research was presented to the second and third author's ethical advisory committees and the project was approved (file number SHW_20_128).

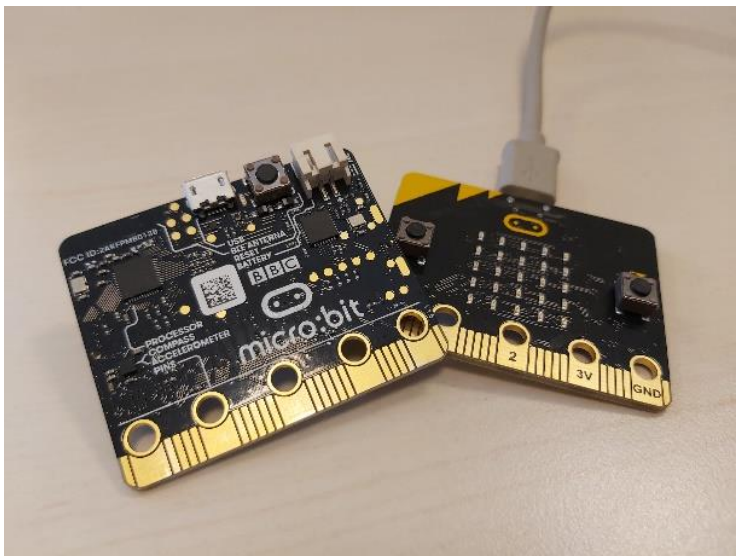
Research task

An assignment was designed in which students had to use an already known programming language (i.e., Microsoft Makecode) in pairs to program a robot (see Figure 7). This robot is powered by a BBC micro:bit (see Figure 8). BBC micro:bit consists of a micro-computer with buttons, an LED matrix as a display, and various sensors. The physical connection pins at the bottom of the micro:bit allow it to be

connected to external motors, LEDs, and other electronic components. Using a Blocks Editor (i.e. Microsoft Makecode), students can program the micro:bit and thus also control the motors that make the robot move. This graphical programming language is designed so blocks can be dragged and attached to each other.

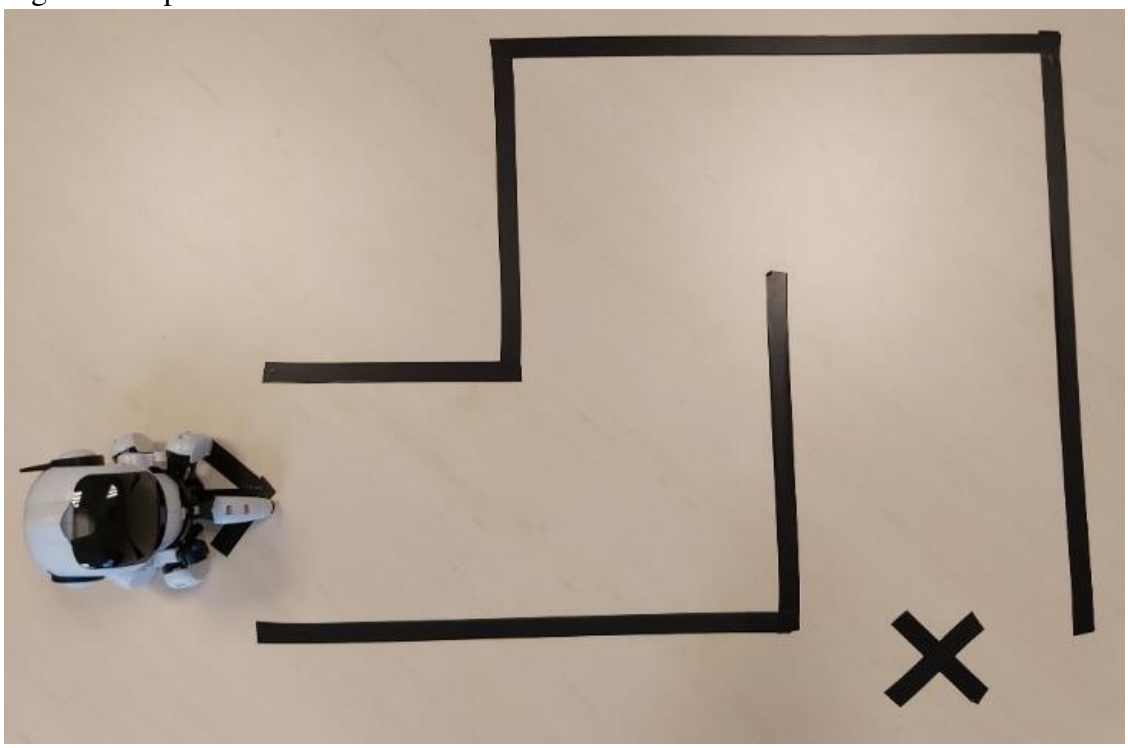
Figure 7. Tobbie II.

Figure 8. BBC micro:bit.



At the start of the assignment, the students were shown a route (Figure 9). This was affixed to the table. The start point was indicated by an arrow and the endpoint by a cross. Each pair was then given 20 minutes to complete the assignment.

Figure 9. Experiment 2 route.



Procedures

Students were invited in pairs. They sat next to each other at a table on which the route had been depicted. On this table was also a laptop and the robot that had to be programmed. To program the robot, the students had to use an online application (i.e., Microsoft Makecode editor), which was available on the laptop. Their coding work was recorded through screen recordings. Audio recording (i.e., Zoom f8n multitrack field recorder) was used to collect verbal data and physical interactions were recorded using an external Ultra-Wide Angle HD Webcam. Everything was later compiled using video editing software (i.e., OpenShot Video Editor and Bandicam) to facilitate the transcribing process. In congruence with Experiment 1, verbal statements were transcribed verbatim, and actions were indicated in the transcripts in brackets.

Analyses

The transcripts were used for qualitative data analysis. For example, the audio, video, and screen recordings collected during the collaborative coding assignments were converted into text. This textual data allowed for further processing with dedicated software (i.e., Nvivo, release 1.3), and allowed the researcher to become familiar with the data. To enable us to distinguish the debugging process from other programming activities, *testing* activities were first mapped out. These were actions where students sent their code to the robot and let it run. Depending on whether this matched their desired outcome, the process that followed was considered debugging or programming. Without testing the program, the output cannot be compared with the intended target and therefore cannot be debugged. The target structure of the model established by Klahr and Carver (1988) proposes this initial step of testing as an essential first phase (see Figure 1). After transcribing and identifying testing events, we coded the text files using two coding schemes. First, we used the scheme developed in Experiment 1 (see

Table 3) to identify different stages of the debugging process (i.e., necessity of debugging, identifying the bug, pinpointing the bug, and correcting the bug).

In addition to coding for the debugging process, the transcripts were also coded for the type of regulation the participants employed during those processes. For this, the adapted scheme of Malmberg et al. (2017) was used (see Table 4).

Subsequently, we examined whether any patterns could be found in the observed debugging processes and the type of regulation used. Initially, this was done using a query matrix that brought together and quantified the coded pieces from the two coding schemes used (i.e., debugging process and regulation type) using software (i.e., Nvivo, release 1.3). The resulting matrix allowed us to identify patterns (e.g., successful versus unsuccessful task completion) and co-occurrences (e.g., necessity for debugging and self-regulated learning) in our qualitative data. As our study is qualitative in nature, each case was examined against these characteristics, and conversations were reported in detail.

Results

The transcripts revealed a total of 161 debugging strategies across 11 of the 15 dyads. Notably, four dyads did not display any testing events and therefore did not undertake any debugging activities. These dyads used their allocated 20 minutes entirely on writing the program without checking how the physical robot would respond to their code until the very end. None of these dyads were able to successfully solve the given task and because no debugging activity was observed in these dyads, they were not included in our further analysis. Across the remaining dyads, all four aspects of debugging (i.e., necessity, identifying, locating, and correcting) and the three types of regulation (i.e., self-regulation, co-regulation, and socially shared regulation) were observed (see Table 7). These strategic actions will be broken down by discussing both

the quantity of actions (i.e., the number of strategic actions identified) as well as the quality of these actions as we can infer from the qualitative data. Finally, we discuss how the debugging processes and regulation types relate to the successful completion of the coding task. Regarding the completion of the coding task and the joint coding success, six dyads completed the task within the given time. The other five dyads managed to program and debug but did not complete the assignment.

A detailed breakdown of identified debugging processes and regulation types for each dyad is outlined in Table 7. This table encompasses the number of coded episodes for each aspect of debugging and the utilization of different types of regulation (SRL, CRL, and SSRL). Dyads vary in their engagement with these processes and types of regulation, leading to a diverse range of outcomes in terms of task completion.

Table 7. Experiment 2: Identified debugging processes and regulation types per dyad.

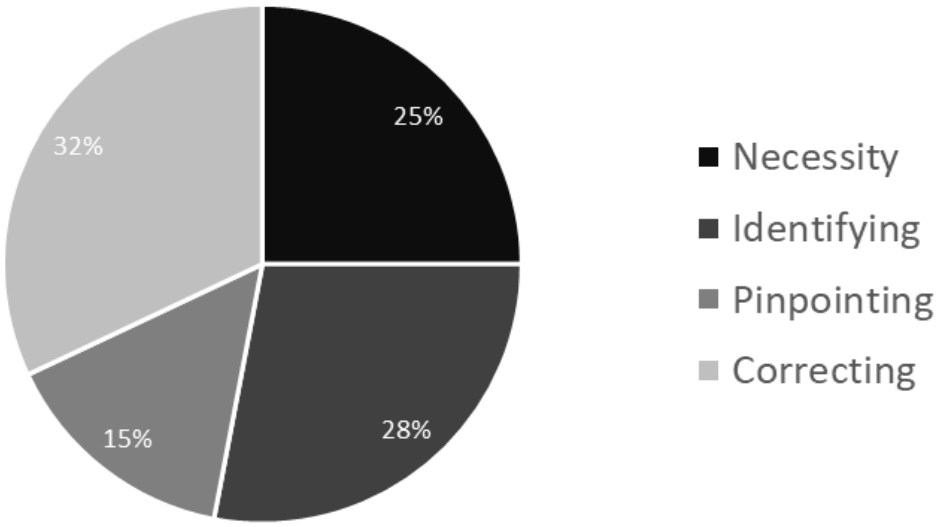
	Number of coded episodes							**Successful task completion**
	Necessity	Identifying	Pinpointing	Correcting	SRL	CRL	SSRL	
Dyad 1	8	5	7	5	10	11	1	No
Dyad 2	6	5	4	6	3	12	6	No
Dyad 3	4	3	2	3	1	1	0	No
Dyad 4	3	2	1	2	3	5	0	No
Dyad 5	0	0	0	0	0	0	0	No
Dyad 6	4	6	4	7	4	13	2	Yes
Dyad 7	0	0	0	0	0	0	0	No
Dyad 8	2	2	1	2	0	6	1	Yes
Dyad 9	3	5	2	4	1	4	8	Yes
Dyad 10	0	0	0	0	0	0	0	No
Dyad 11	0	0	0	0	0	0	0	No
Dyad 12	5	5	1	6	6	3	7	Yes
Dyad 13	1	3	0	6	1	6	1	Yes
Dyad 14	2	6	2	6	4	9	1	Yes
Dyad 15	3	3	0	4	6	4	0	Yes

Note. SRL= self-regulated learning; CRL= co-regulated learning SSRL= socially shared regulated learning

Debugging processes

To address the first research question regarding the debugging processes undertaken during collaborative coding, we focused on four primary debugging phases—necessity, identification, localization, and correction. We systematically coded and quantified these processes. The distribution of these processes across the eleven dyads is depicted in Figure 10, shedding light on the extent to which each process was employed.

Figure 10. Experiment 2: Use of debugging processes across the eleven dyads.



It is paramount to acknowledge that the necessity phase acts as a critical steppingstone for the entire debugging process. As a result, four dyads among the sample refrained from any debugging efforts, having omitted the necessity phase. However, a higher frequency of necessity processes didn't inherently equate to improved coding outcomes. For instance, Dyad 1 and 2 exhibited an emphasis on necessity and localization yet struggled to complete the task within the stipulated time. Successful dyads, conversely, invested more in the identification and correction of the bug. Remarkably, the localization phase was frequently implicit, with dyads transitioning directly from identification to correction. This shift highlighted a deeper grasp of the programming logic.

Further examination through qualitative analysis uncovered that explicit efforts toward bug localization often stem from an inadequate understanding of the programming language. While the Microsoft Makecode editor employs block-based programming, proficiency in program structure and logic is necessary for optimal execution. A telling interaction exemplifies this:

L1: “We need to do something about that…”

(L1 and L2 search between the program blocks)

L1: “We can do light here”

(L2 laughs)

A knowledge gap led certain dyads to allocate more time to localization, hampering their bug correction efforts. As demonstrated in subsequent interactions, some dyads encountered substantial difficulties in pinpointing the bug, resorting to restarting their work multiple times. For instance:

L2: “You just deleted that.” …

L2: “You just deleted everything?”

L1: “I know…”

L1: "Hmmm"

In contrast, successful dyads showcased a more streamlined approach. For instance, Dyad 8, the most successful one, effectively combined all four debugging processes across two cycles. Though quantitatively fewer, their integrated and efficient utilization of these processes paved the way for task completion. This interaction illustrates their adeptness:

L2: “Uh, try?”

L1: “Yeah, okay.”

L2: “Hopefully it will work”. (L2 starts the robot)

L1: “It's going well now.”

L2: “No, wrong.”

L1: “He went sideways”

L2: “He went sideways, but the time was right.”

L1: “Maybe you should see what comes first” (L1 goes over the program)

L2: “We pause for seven seconds the first time. Seven seconds is too much, isn't it?"

L1: “Yes.”

L2: "Couldn't we do 5.5?"

L1: “Or six?”

L2: “Because when we started, he came this far.” (L2 points to the track)

L2: “6.5 seconds maybe?”

L1: "You don't need half a second to get from here to here to here, do you?"

L2: “Okay.” (L2 adjusts the time)

L2: "Shall we try that now?"

L1: "Yeah, that's good."

This dialogue showcases a comprehensive debugging cycle that involved testing, identifying discrepancies, locating the issue, and ultimately rectifying the program.

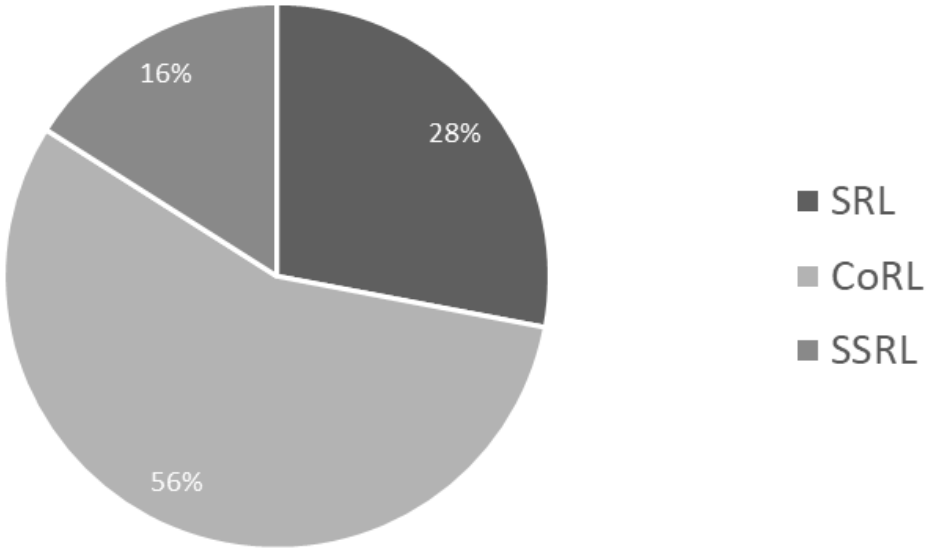
Types of regulation

Turning to the second research question about regulation types employed during debugging, we conducted a thorough coding and quantification of self-regulation (SRL), co-regulation (CRL), and socially shared regulation (SSRL). An overview of the utilization of these types of regulation across the eleven dyads is depicted in Figure 11, illustrating noteworthy disparities in their application.

Overall, co-regulation emerged as the most prevalent form of regulation. However, Dyad 9 deviated from this trend, relying more heavily on SSRL than CRL. Strikingly, seven out of the eleven dyads displayed minimal instances of SSRL. For instance, Dyad 1 only exhibited one instance of SSRL. Dyad 1 also stood out with a higher occurrence of SRL (11 episodes) compared to CRL (10 episodes). Similar to the

debugging processes, variations in the quality of regulation types were noted, influencing the attainment of the task's objective.

Figure 11. Experiment 2: Use of types of regulation across the eleven dyads.



Notably, Dyad 9 successfully completed the task while prominently engaging in SSRL alongside CRL. Their collaboration was characterized by mutual input and action:

L1: “Done?”

L2: “I will download.”

(L1 connects the robot)

L1: “Is it ready? Try again."

L2: “What again?”

(L2 downloads the program and turns on the robot)

L1: “He has to come here. “(L1 points to the course)

L2: “We need to change this one.” (L2 points to the pause blocks on the screen)

However, a robust presence of SSRL did not necessarily guarantee task success. Dyad 2, despite frequent instances of SSRL, fell short of completing the assignment:

L2: “Maybe that should go forward five times.”

L1: “On four occasions it was already here on that line.”

(L1 points to the course)

L1: “At five times he might be there.”

L2: “Then…” (L2 watches the program)

L1: "Then, a 'Tobbie stop' in between."

L2: “We can do that in here.” (L2 adjusts the program)

Dyads 12 and 15 leaned more towards SRL and CRL, successfully completing the task. In these dyads, a significant skill gap led to a stronger participant adopting a leading role, relying more on SRL. Personal and contextual factors also influenced the distribution of regulation types. Dyad 1 showcased high self-efficacy but lacked the necessary skills, paralleling patterns observed in Dyads 12 and 15.

Patterns in debugging processes and types of regulation

Addressing the third and fourth research questions delving into the interplay between debugging processes and types of regulation, and their impact on joint coding success, we found that CRL was the predominant regulation type across all debugging processes, accounting for 50% to 60% of interactions. The alignment or disjunction between regulation types and debugging processes is encapsulated in Table 8. Examining dyads that successfully completed the task and comparing them to those that did not, revealed a significant trend. Successful dyads invested a relatively greater effort in identifying and correcting bugs. This observation becomes evident when examining the frequency of coded episodes, particularly in the context of episodes focused on identifying and rectifying issues versus those centered on pinpointing and addressing underlying causes (see Table 7). Moreover, successful dyads were more inclined to transition to SSRL during these processes (see Table 7).

Table 8. Experiment 2: The intersection of debugging processes with types of regulation.

	Percentage	Mean	Standard Deviation
Necessity SRL	8%	1.18	1.72
Necessity CRL	13%	2.00	1.34
Necessity SSRL	2%	0.36	0.92
Identifying SRL	7%	1.09	1.04
Identifying CRL	17%	2.64	1.43
Identifying SSRL	4%	0.64	1.21
Pinpointing SRL	5%	0.73	0.90
Pinpointing CRL	9%	1.36	1.75
Pinpointing SSRL	2%	0.36	0.92
Correcting SRL	8%	1.27	0.79
Correcting CRL	17%	2.64	1.50
Correcting SSRL	6%	0.91	0.83

Ultimately, these results indicate a strong correlation between successful joint coding and the collaborative interplay between debugging processes and types of regulation. Dyads that achieved the task completion milestone showcased a more integrated approach to debugging, iterating through identification, localization, and correction of bugs. Furthermore, these successful dyads often shifted to socially shared regulation (SSRL) during debugging, suggesting that effective collaboration and mutual effort in navigating programming challenges are closely tied to positive outcomes. While some dyads struggled due to skill gaps, our analysis revealed a more nuanced picture. The engagement in debugging processes and the types of regulation utilized played a pivotal role. Dyads that struggled often spent considerable time on the necessity phase and bug localization, demonstrating challenges in transitioning from identifying issues to effective correction. This observation underscores the complexity of debugging activities in collaborative contexts and hints at potential areas for improvement in collaborative learning.

Discussion

Our discussion will consider the results across both experiments. Experiment 2 builds on the findings of Experiment 1 and taken together they provide a cross-section of novices at different ages. The originality of this work lies in the examination of the interplay between debugging processes, regulation types, and coding outcomes in a collaborative setting. Previous work has demonstrated that debugging is important for individuals, but we investigate here how that is operationalized among a collaborative group. Further, it is known that CRL and SSRL are required in other group problem-solving activities and we endeavour to see if it is also the case here for coding. Building on prior research (Klahr & Carver, 1988; Liu et al., 2017), it explores the relationship between the four debugging processes and successful coding outcomes, while also considering the type of regulation used by novice coders of varying ages. The findings show that the combined and sequential utilization of all debugging processes leads to successful coding and that the type of regulation used depends on multiple factors, such as the dyad members' differential skill levels, domain knowledge, and problem-solving abilities. These must therefore be taken into consideration when designing tasks and composing groups. Moreover, to mediate SSRL, task difficulty must be situated in the zone of proximal development (ZPD, Vygotsky, 1978) of both students. According to Vygotsky (1978), the ZPD is the area where learning takes place most effectively. It is the range of tasks that a learner can accomplish with guidance and support, but not yet independently. Overall, the zone of proximal development is a key concept in educational psychology that highlights the importance of finding the right level of challenge and support for learners to maximize their learning potential. Understanding each team member's ZPD can help to ensure that each team member is challenged appropriately and has the opportunity to learn and grow. The results of this study

provide a fresh perspective on the use of debugging processes and regulation types by novice coders in collaboration and help guide further research and model development.

Although the different debugging processes are defined as separate steps (Klahr & Carver, 1988), they were less explicitly separated in dyads that were more successful in completing the task. Moreover, the testing phase (necessity) was deemed an essential first step to even start any other debugging processes, conforming some sequential relation between the processes. Furthermore, we see similar patterns in the overlap between debugging processes and regulation types. In this respect, the similarities are striking, but the differences, which could be explained by the changed age category and the modified coding assignment, provide unique insights that can guide further research and model development. In what follows we discuss each of these aspects in more detail and formulate implications for both theory and practice.

Debugging processes

We can conclude from Experiment 1 that students undertake all four of the debugging processes (i.e., necessity of debugging, identifying the bug, pinpointing the bug, and correcting the bug). However, if the first step, *necessity* of *debugging* is missing, no further debugging processes can be observed (Klahr & Carver, 1988). For example, 4 out of the 15 dyads in Experiment 2 did not debug their program and could not complete the assignment. Here our findings differ from Experiment 1, where debugging processes were found in all dyads. In Experiment 1 we focused on second-grade students who used a code-a-pillar, a toy robot in the shape of a caterpillar that could be programmed using physical programming blocks. The increased complexity and the use of a programming language (i.e., Microsoft Makecode) may offer an explanation. Computer programming increasingly relies on computational thinking as it increases in complexity, a thought process that uses elements of abstraction, generalization,

decomposition, algorithmic thinking, and debugging (Angeli et al., 2016; Wing, 2006). The four dyads that did not undertake any debugging activities did not divide the assignment into smaller subtasks (i.e., decomposition), which could then be tested and adjusted (i.e., debugging). However, we can conclude that this strategy is not intuitively applied when both digital (e.g., the programming environment) and physical (e.g., the robot) elements are present. When the programming problem only presented itself physically, as in Experiment 1, students automatically proceeded to test and debug their program. As such, the computer-based programming language in which the development environment is separated from the physical output (i.e., the robot) formed an additional barrier that several dyads were unable to overcome. Although block code is semantically simpler than textual coding, understanding the syntax or structure of this language remains crucial. Our observations showed that when students had insufficient mastery of this, they had to work much harder on the necessity of debugging and locating the bug. Dyads more proficient with Microsoft Makecode made comparatively more explicit use of identifying and correcting the bug. The localization of the bug was often implicit, which can again be explained by a better understanding of the programming language.

These findings are in line with previous findings by Klahr and Carver (1988) and Liu et al. (2017), who argue that novice programmers often lack domain knowledge and general problem-solving skills. Both are important in terms of programming, also in a collaborative setting. Moreover, not all students had the necessary knowledge and skills, demonstrating the importance of teaching them. If at least one of the group members had the necessary knowledge and skills, repeatedly applying the four debugging processes in an integrated manner led to better coding outcomes. This ties in

with the findings of Experiment 1, but also illustrates the increased complexity associated with the digital programming of a physical robot.

Types of regulation

Regarding the type of regulation (i.e., SRL, CRL, and SSRL) that dyads used during the debugging processes, we found that dyads who managed to use all three regulation types in a balanced way were more successful in completing the assignment. This supports Hadwin et al. (2011) and Malmberg et al. (2017) findings that argued that the three types of regulation are necessary for a collaborative setting. CRL was by far the most observed type of regulation, but students who adopted SSRL in addition to CRL were more likely to complete the assignment. We did find that the perceived skill level of the participants influenced the type of regulation used. For example, SSRL mainly occurred when both participants found the process challenging and could contribute to the process. When one of the participants was more proficient or self-judged to be more proficient, more SRL was observed. Two dyads (i.e., 12 and 15) showed that this was not always disadvantageous. They also completed the assignment within the set time because one student took the lead.

Where domain knowledge and problem-solving skills of an individual group member proved to be decisive for the use of certain debugging processes, we note here that the ratio of skills between the group members influences the type of regulation used. For example, SSRL mainly occurred when the assignment presented a challenge to both group members and each group member could meaningfully contribute to its completion. However, these aspects were not addressed in Experiment 1, which also underlines the importance of contextual factors. Eighth-grade students may be shaped more by previous experiences and more subjective to social factors.

Patterns in debugging processes and types of regulation

When analyzing the overlap between debugging processes and the type of regulation used, we noticed complex patterns. In dyads who successfully completed the assignment, SSRL was used proportionally more when identifying and correcting the bug. When pinpointing, hardly any SSRL was detected. In dyads that were less successful in programming their robot, we observed much fewer episodes of SSRL. Even if this was the case, it mainly happened during locating actions. This leads us to suspect that these dyads had trouble finding the bug and lacked knowledge regarding the programming language. This observation in Experiment 2 raises questions about the level of programming skill required for successful collaborative coding and the role of collaborative learning environments in bridging skill gaps. While successful dyads integrated debugging processes and demonstrated effective collaboration, the less successful ones could benefit from targeted interventions to enhance their programming competencies.

The observed patterns in both processes and types of regulation show that debugging is a complex and dynamic process (Ahn et al., 2017; Klahr & Carver, 1988), even with a relatively simple coding task performed by eighth-grade students. As the type of regulation used depends on the perceived difficulty of the task, our observations indicate that it can differ per debugging process. This finding conforms to Järvelä and Hadwin's (2013) framework that describes these process-based differences in regulation as task dependencies. However, the perception of skills, as well as contextual factors, are not included in this model.

Limitations

This study is based on two small samples of students from each grade and in each experiment students all came from the same school. Our findings, therefore, are not generalizable.

One of the main limitations of our study is that both experiments relied primarily on the verbal statements of students while solving problems to measure their debugging and regulation processes. The extent to which students verbally externalize these processes might depend on the personal characteristics of the students which have not been taken into account in this study. Future research could take such personal characteristics more into account and could also aim to triangulate the verbal statements with process data coming (e.g., the log files of their coding in Experiment 2). While we have presented evidence here that more successful groups employ all different types of regulation (i.e., SRL, CRL, and SSRL), the nature of the individuals within the groups may also mediate the relation between regulation and coding outcomes and needs to be further investigated.

A relevant difference to note in the findings is that debugging was coded throughout the entirety of children's interactions with the robot in Experiment 1. In Experiment 2, debugging was captured when students sent the code to the robot for testing and when they were evaluating what happened. That is why there are more total instances of debugging for the smaller sample in Experiment 1 than in the larger sample of Experiment 2. Despite this difference, the pie charts presented show very similar rates of each type of debugging between the two age groups. As noted earlier, the decision for what to include as coded activity stemmed from the nature of working with tangible block coding versus block coding on screen.

Implications

The results from this cross-sectional study hold valuable information for research on the advancement of debugging techniques and collaborative programming as a whole. The findings show that novice programmers possess prior understanding (e.g., problem solving abilities) that prompts them to utilize debugging methods even before receiving formal training. Despite the importance placed on teamwork in computer science education, this study highlights specific areas where strategy training could effectively improve collaborative programming efforts. The observed variations in success rates and debugging strategies among dyads highlight the importance of acknowledging and accommodating diverse skill levels within collaborative learning settings. Collaborative coding tasks should be designed and scaffolded in a way that fosters effective collaboration between students with varying programming abilities. This might involve implementing strategies to encourage peer mentoring, shared problem-solving, and the development of complementary skills. To mediate SSRL, the task difficulty must be appropriately aligned with the zone of proximal development (Vygotsky, 1978) for both students.

Furthermore, the influence of programming skills on collaborative coding outcomes underscores the need for a balanced approach that integrates technical proficiency with collaborative problem-solving. Emphasizing the development of programming skills alongside teamwork and communication abilities could lead to more successful collaborative coding experiences (Sharma et al., 2019). The findings of this study challenge the notion that programming and debugging can be reduced to a set of clearly defined consecutive steps. Instead, it highlights the dynamic nature of coding and debugging processes and the importance of including metacognitive and regulatory elements in debugging models.

The examination of debugging strategies and their correlation with collaborative success offers educators valuable insights into optimizing the teaching of debugging skills and fostering strategic thinking. A study with older students suggests that understanding of the types of regulation employed during debugging processes enables instructors to customize their pedagogical strategies, providing targeted support to students as they navigate challenges, improve problem-solving methodologies, and enrich their overall learning journey (Emara et al., 2021). By explicitly instructing students in the art of effective debugging and self-regulation, educators equip them to confront coding complexities with heightened efficiency and collaborative prowess. Additionally, this study emphasizes the importance of accounting for contextual variables when deciphering the intricacies of programming and debugging practices within collaborative contexts.

With regard to the socially shared regulation learning model, the data from this study suggest that co-regulation (CRL) may play a key role in facilitating the development of shared regulation strategies, particularly for novice programmers.

Our observational study provides a foundation for further research exploring the intersection of programming skills, collaborative dynamics, and learning outcomes. Further investigation could delve into the impact of different instructional methods, peer interactions, and technological tools on collaborative coding success and skill development. A mixed-method design, including qualitative and quantitative methods, could improve data triangulation and provide a better understanding of patterns. This can be achieved by supplementing observations with log data, questionnaires, and interviews to explore differences in skill level, personality, motivation, perception, and task difficulty.

In conclusion, the implications of this study extend beyond the presented results, offering insights into the optimization of collaborative learning environments, the cultivation of programming skills, and the design of effective pedagogical strategies. These implications hold the potential to enhance both the learning experiences and outcomes of students engaged in collaborative coding tasks.

References

Ahn, J. H., Mao, Y., Sung, W., & Black, J. B. (2017, March). Supporting Debugging Skills: Using Embodied Instructions in Children's Programming Education. In *Society for Information Technology & Teacher Education International Conference* (pp. 19-26). Association for the Advancement of Computing in Education (AACE).

Alexander, P. A., & Judy, J. E. (1988). The interaction of domain-specific and strategic knowledge in academic performance. *Review of Educational Research*, *58*(4), 375-404.

Allal, L. (2010). Assessment and the Regulation of Learning. *International Encyclopedia of Education,* 348-352.

Angeli, C., Voogt, J., Fluck, A.E., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 Computational Thinking Curriculum Framework: Implications for Teacher Knowledge. *Journal of. Educational Technology & Society, 19*, 47-57.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). Developing Computational Thinking in Compulsory Education-Implications for Policy and Practice. Luxembourg: Publications Office of the European Union.

Bruning, R., Schraw, G., & Norby, M. (2010). *Cognitive Psychology and Instruction*. Pearson, New York, NY, USA.

Bull G., Garofalo J. & Hguyen N. R. (2020). Thinking about computational thinking, *Journal of Digital Learning in Teacher Education*, 36:1, 6-18.

Chalmers, A. F. (1982). Epidemiology and the scientific method. *International Journal of Health Services*, *12*, 659-666.

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012, February). Social coding in GitHub: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work* (pp. 1277-1286). ACM.

DiDonato, N. (2013). Effective self- and co-regulation in collaborative learning groups: An analysis of how students regulate problem solving of authentic interdisciplinary tasks. *Instructional Science, 41*, 25-47.

Dignath, C., & Buettner, G. (2008). Components of fostering self-regulated learning among students. A meta-analysis on intervention studies at primary and secondary school level. *Metacognition and Learning, 3*, 231-264.

Dinsmore, D. L. (2017). Towards a dynamic, multidimensional model of strategic processing. *Educational Psychology Review, 29,* 235-268. doi: 10.1007/s10648-017-9407-5

Duncan, C., Bell, T., & Tanimoto, S. (2014). Should your 8-year-old learn coding? *Proceedings of the 9th Workshop in Primary and Secondary Computing Education on - WiPSCE '14*.

Emara, M., Grover, S., Hutchins, N., Biswas, G., & Snyder, C. (2020). Examining Students' Debugging and Regulation Processes During Collaborative Computational Modeling in Science. In Gresalfi, M. and Horn, I. S. (Eds.), *The Interdisciplinarity of the Learning Sciences, 14th International Conference of the Learning Sciences (ICLS) 2020*, Volume 3 (pp. 1325-1332). Nashville, Tennessee

Emara, M., Hutchins, N. M., Grover, S., Snyder, C., & Biswas, G. (2021). Examining student regulation of collaborative, computational, problem-solving processes in open-ended learning environments. *Journal of Learning Analytics*, *8*(1), 49-74.

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.

Grover, S., & Pea, R. (2013). Computational Thinking in K–12. *Educational Researcher, 42*(1), 38-43.

Hadwin, A. F., Järvelä, S., & Miller, M. (2011). Self-regulated, co-regulated, and socially shared regulation of learning. In B. Zimmerman & D. Schunk (Eds.), *Handbook of self-regulation of learning and performance* (pp. 65-84). New York: Routledge.

Järvelä, S., & Hadwin, A. F. (2013). New Frontiers: Regulating Learning in CSCL. *Educational Psychologist, 48*(1), 25-39.

Kafai, Y., & Vasudevan, V. (2015, June). Hi-Lo tech games: crafting, coding and collaboration of augmented board games by high school youth. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 130-139). ACM.

Kallia, M., & Cutts, Q. (2022). Conceptual development in early-years computing education: A grounded cognition and action based conceptual framework. *Computer Science Education*. Advance online publication. https://doi.org/10.1080/08993408.2022.2140527

Klahr, D., & Carver, S. M. (1988). Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology*, *20*, 362-404.

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

Liu, Z., Zhi, R., Hicks, A., & Barnes, T. (2017). Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education, 27*(1), 1-29.

Malmberg, J., Järvelä, S., & Järvenoja, H. (2017). Capturing temporal and sequential patterns of self-, co-, and socially shared regulation in the context of collaborative learning. *Contemporary Educational Psychology*, *49*, 160-174.

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., Fitzgerald, S., Hanks, B., & Mccauley, R. (2010). Pair debugging. *Proceedings of the Sixth International Workshop on Computing Education Research - ICER '10.*

Neuendorf, K. A. (2017). *The content analysis guidebook*. Sage Publications

Panadero, E. (2017). A Review of Self-regulated Learning: Six Models and Four Directions for Research. *Frontiers in Psychology, 8*.

Panadero, E, & Järvelä, S. (2015). Socially shared regulation of learning: A review. *European Psychologist, 20*(3), 190-203. https://doi.org/10.1027/1016-9040/a000226

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. NY: Basic Books Inc.

Potters, O. (2014). Regulatieprocessen Tijdens Samenwerkend Leren [Unpublished Master's Thesis]. Utrecht University

Saldaña, J. (2015). *The coding manual for qualitative researchers*. Sage Publications.

Scherer, R. (2016). Learning from the Past–The Need for Empirical Evidence on the Transfer Effects of Computer Programming Skills. *Frontiers in Psychology, 7*

Sharma, K., Papavlasopoulou, S., & Giannakos, M. (2019). Coding games and robots to enhance computational thinking: How collaboration and engagement moderate children's attitudes? *International Journal of Child-Computer Interaction*, *21*, 65-76. https://doi.org/10.1016/j.ijcci.2019.04.004

Tang, K., Chou, T., & Tsai, C. (2019). A Content Analysis of Computational Thinking Research: An International Publication Trends and Research Typology. *The Asia-Pacific Education Researcher, 29*(1), 9-19.

Vygotsky, L. S. (1978). Mind in society: The development of higher psychological processes. Cambridge, MA: Harvard University Press.

Villamor, M., Paredes, Y. V., Samaco, J. D., Cortez, J. F., Martinez, J., & Rodrigo, M. M. (2017). Assessing the Collaboration Quality in the Pair Program Tracing and Debugging Eye-Tracking Experiment. *Lecture Notes in Computer Science Artificial Intelligence in Education,* 574-577.

Vlaamse regering (2019). *Decreet betreffende de onderwijsdoelen voor de eerste graad van het secundair onderwijs*. Belgisch staatsblad, publicatiedatum 26 april 2019.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

Winne, P. H. (1995). Inherent details in self-regulated learning. *Educational Psychologist*, *30*(4), 173-187.

World Economic Forum (2016). The future of jobs: Employment, skills and workforce strategy for the fourth industrial revolution. *Global Challenge Insight Report.*

Zimmerman, B. J. (2000). Attaining Self-Regulation: A Social Cognitive Perspective. In M. Boekaerts, P. R. Pintrich, & M. Zeidner (Eds.), *Handbook of Self-Regulation* (pp. 13-39). San Diego, CA: Academic Press.