The Journal of Systems & Software 183 (2022) 111107

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

How secondary school girls perceive Computational Thinking practices through collaborative programming with the micro:bit ☆

Mojtaba Shahin [a,*], Christabel Gonsalvez [a], Jon Whittle [b], Chunyang Chen [a], Li Li [a], Xin Xia [a]

[a] Faculty of Information Technology, Monash University, Australia
[b] CSIRO's Data61, Clayton, Australia

ARTICLE INFO

Article history:
Received 27 February 2021
Received in revised form 1 October 2021
Accepted 2 October 2021
Available online 8 October 2021

Keywords:
Computational thinking practices
Girls
Education
K-12
Programming

ABSTRACT

Computational Thinking (CT) has been investigated from different perspectives. This research aims to investigate how secondary school girls perceive CT practices – the problem-solving practices that students apply while they are engaged in programming – when using the micro:bit device in a collaborative setting. This study also explores the collaborative programming process of secondary school girls with the micro:bit device. We conducted mixed-methods research with 203 secondary school girls (in the state of Victoria, Australia) and 31 mentors attending a girls-only CT program (OzGirlsCT program). The girls were grouped into 52 teams and collaboratively developed computational solutions around realistic, important problems to them and their communities. We distributed two surveys (with 193 responses each) to the girls. Further, we surveyed the mentors (with 31 responses) who monitored the girls, and collected their observation reports on their teams Our study indicates that the girls found "debugging" the most difficult type of CT practice to apply, while collaborative practices of CT were the easiest. We found that prior coding experience significantly reduced the difficulty level of only one CT practice - "debugging". Our study also identified six challenges the girls faced and six best practices they adopted when working on their computational solutions.

1. Introduction

Computational Thinking (CT) has been widely researched due to its benefits for public services (e.g., education and healthcare), business sectors (e.g., financial markets), and society generally (Wing, 2014; Snalune, 2015). The current research on CT has mostly focused on integrating CT into academic disciplines (e.g., biology Sengupta et al., 2013 and mathematics Weintrop et al., 2016), the K-12 curriculum (e.g., Sengupta et al., 2013; Yadav et al., 2016), and developing programming environments and tools to promote CT skills (e.g., Grover and Pea, 2013; Lye and Koh, 2014). Some research also examines and measures the learning outcomes of learners in three dimensions of CT: computational concepts, computational practices, and computational perspectives (Lye and Koh, 2014; Kong, 2019; Brennan and Resnick, 2012). Others investigated the (frequency of) learning barriers (e.g., programming syntax, debugging) that students encounter in programming courses in academic settings and coding clubs (e.g., Dorn et al., 2018; Aivaloglou and Hermans, 2019). According to Lye and Koh (2014), most of the reported research in this area has focused on assessing the learning outcomes in terms of computational concepts such as the works done by Sherman and Martin (2015) and Grover et al. (2015), but there are very few studies examining and assessing the learners' ability to develop and apply computational practices (Rodriguez et al., 2017), and computational perspectives (Denner et al., 2014). Reviews (Zhang and Nouri, 2019; Lye and Koh, 2014; Kong, 2019) have emphasised that there is a need for more empirical research to examine computational practices and perspectives in K-12 settings. This can be mainly justified by the fact that computational practices and perspectives are bigger contributors than computational concepts in achieving the main goal of introducing CT through programming in K-12 settings, providing students with a set of skills (e.g., problem-solving skill) that they can leverage in their daily lives (Lye and Koh, 2014; Resnick et al., 2009; Wing, 2006). Furthermore, the existence of different definitions of CT in the literature implies that little consensus exists regarding the nature and the relative importance of particular CT practices (Voogt et al., 2015; Shute et al., 2017). For example, young learners may individually develop one CT practice (e.g., debugging) within a particular period of time (Wohl et al., 2015), whilst other CT practices such as collaborative problem-solving require mutual engagement and develop gradually over time, which might also

☆ Editor: Alexander Serebrenik.

* Corresponding author.
E-mail addresses: mojtaba.shahin@monash.edu (M. Shahin), chris.gonsalvez@monash.edu (C. Gonsalvez), jon.whittle@data61.csiro.au (J. Whittle), chunyang.chen@monash.edu (C. Chen), li.li@monash.edu (L. Li), xin.xia@monash.edu (X. Xia).

https://doi.org/10.1016/j.jss.2021.111107
0164-1212/

be influenced by broader sociocultural factors (Fawcett and Garton, 2005). This also indicates that learners experience different levels of difficulty when developing and applying particular CT practices (Wohl et al., 2015; Atmatzidou and Demetriadis, 2016). Despite the ongoing call to research CT practices, there have been few rigorous empirical investigations into CT practices and even less (if any) into the difficulty level of CT practices from the learners' perspective in K-12 settings (Kong, 2019), especially from the viewpoint of girls.

The importance of CT practices in daily life and a wide range of disciplines and professions has motivated educational researchers, governments, and practitioners to attempt to achieve "CT for all" goal (Angevine et al., 2017; Angeli and Valanides, 2020). A growing number of interventions in the forms of CT programs, coding clubs, or computer science classes have been designed and delivered to move toward full participation in a computational world. Although such interventions target both men and women, lower representation by women continues to be an issue in the educational and professional worlds of computing and STEM fields (Grover and Pea, 2013; Wang and Degol, 2017; Botella et al., 2019). While an important body of literature (e.g., Atmatzidou and Demetriadis, 2016; Durak and Saritepeci, 2018; Ardito et al., 2020) shows that there is overwhelmingly more similarity than difference between girls and boys in terms of skills, competence, and achievement in computing, several factors deter girls from pursuing, choosing, and persisting in computing education and career paths (Ashcraft et al., 2012). These factors are varied but can be generally categorised into *psychological factors* (e.g., gender stereotypes such as "girls lack computing skills" or "people in the computing arena are geeky"), *social factors* (e.g., the influence of parents and peers and a lack of female role models), and *structural factors* (e.g., exposing girls to computing environments and curriculums and learning pedagogies that are uncomfortable for them) (Ashcraft et al., 2012; Barker and Aspray, 2006; Vitores and Gil-Juárez, 2016; Lang et al., 2020). The above factors can significantly affect girls' perceptions, confidence, and interests in computing (Ashcraft et al., 2012; Vitores and Gil-Juárez, 2016). A promising approach to attempt to address this challenge is girls-only after school interventions (Angevine et al., 2017; Ashcraft et al., 2012; Seneviratne, 2017; Brady et al., 2017; Spieler et al., 2019). In contrast to mixed-gender computing education programs, girls-only education programs provide safer and more comfortable environments for girls to develop intention and a positive view towards computing and boost their confidence in computing (Ashcraft et al., 2012; Barker and Aspray, 2006; Lang et al., 2020; Boddington and Barakat, 2018). The question of *when* and *how* to present computational opportunities to women to increase participation is of interest. Research (Wang and Degol, 2017; Sadler et al., 2012; Gabay-Egozi et al., 2014) suggests that the optimal time for cultivating girls' interest in STEM and computing is during late childhood and early adolescence, as they have more authority to take the opportunities they are interested in Pinkard et al. (2017). Furthermore, it is argued that exposure of women to developing computational ideas and solutions to realistic problems in social and collaborative contexts can be crucial motivating factors for them to pursue a STEM and/or computing education or career in the future (Ashcraft et al., 2012; Brady et al., 2017; Tissenbaum et al., 2019; Kesar, 2017).

This work aims to understand how CT practices are perceived by secondary school girls (14–6 years old) when developing and implementing computational ideas and solutions with the micro:bit device,[1] in a problem-based learning context in a collaborative setting (Hmelo-Silver, 2004). More specifically, our research (a) investigates how secondary school girls perceive the difficulty level of CT practices including planning, decomposition, abstraction, generalisation, algorithm, testing and debugging, and collaboration, which are more likely to emerge and be evaluated while students engage in programming activities; (b) investigates the impact of secondary school girls' prior coding experience on the perceived difficulty level of CT practices; (c) describes the challenges and barriers that secondary school girls experience when developing and implementing computational ideas and solutions; and (d) identifies the practices and techniques that secondary school girls learn, develop, and apply to overcome these challenges and barriers.

To that end, we propose the following research questions:

RQ1. What are the perceptions of secondary school girls on the difficulty level of CT practices when doing collaborative programming with the micro:bit?

RQ2. Is there a relationship between secondary school girls' prior coding experience and their perceptions of the difficulty of CT practices?

RQ3. What challenges do secondary school girls face when collaboratively implementing computational ideas with the micro:bit?

RQ4. What practices do secondary school girls employ to overcome these challenges?

We performed mixed-methods research, which collected data from the participants of a girls-only CT program (i.e., it is referred to as the OzGirlsCT program in this paper) to answer our research questions. (a) We distributed two surveys (with 193 valid responses each) to 203 secondary school girls who participated in the OzGirlsCT program. (b) We conducted another survey with 31 valid responses from 31 mentors who guided and closely monitored the girls during the OzGirlsCT program. (3) We asked the mentors to provide their observations (i.e., in total, 52 observation reports) on the work habits, behaviours, and experiences of the girls in their teams during the OzGirlsCT program.

The main findings of this study are: (1) our participants (i.e., secondary school girls and mentors) quantitatively indicate that "*debugging*" is the most difficult type of CT practice to apply, followed by "abstraction". (2) Collaborative practices of CT are the easiest practices to apply. (3) Prior knowledge and experience of coding can significantly reduce the difficulty level of "*debugging*"; (4) The challenges that the secondary school girls face when developing and implementing computational solutions with the micro:bit can be attributed to "*incorporating idea into the micro:bit*", "*code debugging*", "*code complexity*", "*the micro:bit limitations*", "*personality traits*", and "*coding experience*"; and (5) the main practices employed by the girls to overcome the challenges are "*feedback-driven development*", "*establishing a collaborative and supportive culture within the team*", "*simple design, better code*", "*predictive thinking*", "*prioritising quantity over quality*", and "*leveraging external resources*".

The key contributions of this study are summarised as follows:

- A relatively large-scale study that employs both quantitative and qualitative analyses to understand how secondary school girls perceive CT practices through collaborative programming with the micro:bit;
- A better understanding of the difficulty level of CT practices;
- An empirical investigation into the collaborative programming process of secondary school girls in their early efforts in programming;
- Concrete and actionable implications for educational researchers, practitioners, and policymakers.

This paper is organised as follows: In Section 2, we describe the background and related work. Section 3 details our research method, and our findings are reported in Section 4. Our discussion and reflection on the findings are presented in Section 5. Finally, Section 6 summarises our study.

[1] https://microbit.org/.

2. Background and related work

This section reports some background for the research presented in this study, along with a brief discussion of the related studies.

2.1. Computational thinking definition and scope

Computational Thinking (CT) is increasingly acknowledged as a set of fundamental skills to nurture, equip, and inspire the next generation for the workforce of the digital era (Bocconi et al., 2016; National Research Council, 2010; Kong et al., 2019). The idea behind CT was introduced by Papert (1980) and then popularised by Wing (2006). Despite being researched for almost two decades, there is no single, overarching definition of CT, and little consensus exists as to what skills and competencies constitute CT (Zhang and Nouri, 2019; Brennan and Resnick, 2012; Cuny et al., 2010). Whilst Aho (2012) conceptualises CT as "*the thought processes involved in formulating problems so their solution can be represented as computational steps and algorithms*", CT is viewed by Cuny et al. (2010) as a skill set which everyone should develop to formulate and solve problems like a computer scientist. The promising benefits of CT have stimulated widespread interest among educational researchers, practitioners, and policymakers to integrate and implement CT into STEM (Science, Technology, Engineering, and Mathematics) curricula and K-12 education (Lye and Koh, 2014; K-12 Computer Science Framework Steering Committee, 2016; Barr and Stephenson, 2011). CT is referenced as a core component of STEM disciplines, in particular, the Computer Science (CS) discipline (Grover and Pea, 2013; Angevine et al., 2017). In addition to the efforts to define CT, Brennan and Resnick (2012) developed a framework for operationalising CT in K-12 education. The framework has three dimensions: computational concepts (i.e., the concepts such as *sequences* and *conditional statements* that learners use in their program), computational practices (i.e., the problem-solving practices such as *decomposition* and *debugging* that learners develop and apply while they are engaged in programming), and computational perspectives (i.e., this dimension includes the perspectives that are formed by learners about themselves and the world around them).

2.2. Computational thinking practices and computing teaching

As noted in Section 2.1, there is no agreement on what practices matter in CT. Some researchers use different terminology, such as CT skills or CT competencies, to refer to CT practices. Brennan and Resnick (2012) suggest that the four core practices in CT are "abstracting and modularising", "reusing and remixing", "testing and debugging", and "being incremental and iterative". Grover and Pea (2013) refer to a broader list of skills than of Brennan and Resnick (2012) as CT skills: "abstractions and pattern generalisations", "systematic processing of information", "symbol systems and representations", "algorithmic notions of flow of control", "structured problem decomposition (modularising)", "iterative, recursive, and parallel thinking", "conditional logic", "efficiency and performance constraints", and "debugging and systematic error detection". On the other hand, Korkmaz et al. (2017) consider creativity, critical thinking, and cooperativity as CT practices. Still, others refer to communication and working effectively in teams as key practices in CT (Grover et al., 2015; Berl and Lee, 2011; Astrachan et al., 2011).

Researchers have investigated the process of developing and acquiring CT skills from different perspectives such as age, gender, and pedagogical strategies. Atmatzidou and Demetriadis 2016 explored how generalisation, algorithms, abstraction, decomposition, and modularity skills are developed in the context of robotics among students grouped based on age and gender. Whilst the study found that age and gender did not impact the development level of CT skills, it has shown that girls needed more time and effort to achieve the same skill level. Similarly, Durak and Saritepeci (2018) indicated that the gender of students did not affect their CT skill levels. Doleck et al. (2017) empirically showed that there was no significant association between academic performance and four CT skills, including creativity, critical thinking, algorithmic thinking, and problem-solving, but their study only found that cooperativity as a CT skill had a negative relationship with academic performance. In another study (Durak and Saritepeci, 2018), it was found that these four CT skills could be highly predicted by academic success in mathematics classes. From the teacher's perspective, Günbatar (2019) evaluated the same set of CT skills and found that in-service teachers were significantly better in the development of this set of CT skills compared to pre-service teachers, except for problem-solving. Lewis (2012) conducted a case study to understand the behaviour of sixth-grade students in the debugging process. He observed that a key competence in debugging is identifying and paying attention to the important elements of code state. He also found that having domain knowledge and such competence mediates the debugging process.

Aivaloglou and Hermans (2019) studied teachers' perspectives about code clubs and revealed that debugging and abstract thinking were the most frequent programming learning barriers for students attending code clubs. The study (Aivaloglou and Hermans, 2019) also found that girls were better than boys in collaboration and communication skills. In another study (Dorn et al., 2018), Dorn et al. collected data about learning barriers in programming from teachers and computer science first-year students before the first lecture started. It was found that "way of thinking" (e.g., abstract thinking, complex thinking, logical thinking) was the most common barrier perceived by the students. At the same time, the teachers believed that "programming language/syntax" and "diligence/commitment/stubbornness" were the most common challenges. However, both groups indicated that debugging was difficult but less frequently mentioned by both groups.

Some studies (e.g., Alvarado et al., 2018; Wilcox and Lionelle, 2018; Kirkpatrick and Mayfield, 2017) indicated the positive impact of prior coding/computing experience on students' performance in computer science courses. Alvarado et al. (2018) observed this positive impact on student grades in introductory and advanced courses in computer science, revealing students with pre-college computing experience performed significantly better than their peers with less or without experience in these courses. Wilcox and Lionelle (2018) achieved the same findings for the introductory computer science course but found that the impact of computing experience in advanced courses was gradually diminished. On the other hand, Durak and Saritepeci (2018) found that the experience of secondary and high school students using ICT did not impact their CT skill levels.

Although CT skills are usually introduced and evaluated through computer programming activities in schools, several studies have claimed that the unplugged approach (i.e., when digital devices are not used) is an effective approach for this purpose (Rodriguez et al., 2017; Brackmann et al., 2017; Feldhausen et al., 2018). By investigating pattern recognition, algorithmic design, decomposition, and abstraction, Brackmann et al. (2017) showed that learners who participated in the unplugged activities developed this set of CT practices significantly more than those who were not engaged with these unplugged activities.

Our work is different from the existing studies: (1) our findings come from surveying 193 secondary school girls and a survey and an observation report completed by 31 mentors rather than

only *school students* (Lewis, 2012), *teachers* (Aivaloglou and Hermans, 2019), or *university students* (Dorn et al., 2018; Alvarado et al., 2018; Wilcox and Lionelle, 2018). This study (Dorn et al., 2018) collected data from both students and teachers; however, it targeted first-year university students, not secondary school students. (2) The studies (Dorn et al., 2018; Aivaloglou and Hermans, 2019; Lewis, 2012) either focused on *one CT practice* or *a few particular CT practices* (e.g., debugging, abstract thinking) and indicated that how frequent they are reported as a learning barrier in programming among many other types of learning barriers (e.g., programming syntax). Our study, however, examines the difficulty level of 12 CT practices (**RQ1**). (3) The studies (Alvarado et al., 2018; Wilcox and Lionelle, 2018) focused on the impact of prior coding/computing experience on *student grades* in computer science courses. In contrast, our work assesses the impact of coding experience on the difficulty level of 12 CT practices (**RQ2**). (4) Finally, our study provides a rigorous exploration and analysis of the programming process of secondary school girls with the micro:bit device from the perspective of the challenges that they may face in their early programming efforts and the best practices that they develop and apply to overcome these challenges (**RQ3** and **RQ3**).

For the sake of consistency and readability, we adopted the term *CT practice*, as suggested by Brennan and Resnick (2012), instead of *CT skill* or *CT competence* in this study.

2.3. Problem-based learning

Recently, researchers and practitioners have shown more interest in designing learning environments that provide young learners an opportunity to learn and develop CT concepts and practices through working on problems that are *authentic* and *relevant to their lives* and *their interests* (Lye and Koh, 2014; Tissenbaum et al., 2019). This consideration is also stressed by researchers from other disciplines (Soppe et al., 2005; Hung, 2006). Tissenbaum et al. (2019) referred to this trend as *computational action*, which can lead to more intellectual engagement of the learner in problem-solving activities (Kafai and Resnick, 2012). Hsu et al. (2018) revealed that CT can fit into many learning strategies such as "problem-based learning", "project-based learning", and "game-based learning", we found a strong synergy between problem-based learning (PBL) and this new trend in CT, as PBL deals with ill-structured, realistic problems that have a significant impact on learners' lives, and chose to use this learning strategy for the OzGirlsCT program.

PBL is a type of experiential learning in which the learning process of students is facilitated by teachers (Hmelo-Silver, 2004). Given the increasing importance of the ability to both identify problems and provide solutions to solve the problems in the 21st century, PBL as an instructional approach situates learning in real-world problems and offers the potential to help learners identify real-life, ill-structured problems, develop solutions, and construct knowledge (Savery, 2015). Learners work in collaborative groups and take responsibility for their learning in the PBL approach. During the learning process, learners are responsible for identifying the learning issues (i.e., their insufficiencies and strengths in resolving a realistic, ill-structured problem). They have to discover what new knowledge they need to acquire to fix the problem (i.e., known as "self-directed learning"). This necessitates an extensive reflection on the knowledge being constructed and the effectiveness of the solutions proposed and employed (Bereiter and Scardamalia, 1989). In PBL, teachers (also known as tutors) only facilitate and activate the collaborative learning process and do not provide the information related to the problem (Hmelo-Silver, 2004; Savery, 2015). Hmelo-Silver (2004) asserts that PBL can promote the construction of flexible knowledge and the development of skills such as problem-solving, collaboration, critical thinking, and self-directed.

2.4. Micro:bit

Programming environments and languages for education undergo tremendous change, and every year new environments and tools emerge. This compels educational researchers to investigate if and how the emerging learning environments and tools affect the learning outcomes of learners and the difficulties that they may experience when developing certain CT skills (Zhang and Nouri, 2019; Weintrop and Wilensky, 2018). The micro:bit is a "RAM-based programmable Internet of Things (IoT) device" created by the BBC to train children about programming and preliminary computing principles (Ball et al., 2016; Videnovik et al., 2018). The micro:bit was released formally in 2016 and has been used by more than 20 million children in 60 countries (Micro:bit Educational Foundation, 2021). It is a low-cost, pocket-size device (4 × 5 cm) with an ARM Cortex-M0 Processor to execute the programs. The micro:bit device includes various features such as a 256 KB flash memory, an accelerometer, 16 KB RAM, two programable buttons, one reset button, and 25 programmable LEDs organised in a 5 × 5 grid (See Fig. 1) (Micro:bit Educational Foundation, 2021). There are three options to write a program with the micro:bit: Blocks, JavaScript, and Python. Users can create programs using a block-based programming language, dragging and dropping blocks in a logical order. Microsoft MakeCode[2] and MicroPython[3] are two official text-based code editors of the micro:bit. The former provides users with JavaScript programming experience, and the latter supports Python. These features characterise the micro:bit as a hybrid block/text programming environment (Weintrop and Wilensky, 2018), in which users can switch back-and-forth between the block-based and text-based editors. A wide range of educational projects can be implemented using the micro:bit device, ranging from developing classical and interactive games (e.g., rock, paper, scissors game), to building prototypes to track, monitor, and simulate objects in environments (e.g., a prototype to monitor how climate change affects animals), to helping people with special needs (e.g., a prototype tool to support autistic people in communicating with others) (Micro:bit Educational Foundation, 2021; Kelion, 2021).

Most of the previous investigations of the micro:bit device are focused on fostering learners' enthusiasm and interest in computing and programming (Videnovik et al., 2018; Fessard et al., 2019), but no research has explored how learners develop and apply CT practices with the micro:bit as a hybrid block/text programming environment. In this study, we leveraged the micro:bit device and Microsoft MakeCode as our introductory programming environment for two reasons: (1) Whilst the micro:bit provides learners with hands-on experience of coding similar to dominating programming environments around CT (e.g., Scratch[4]), it also engages novice users with ubiquitous computing through providing insights into embedded systems and enabling them to understand how hardware and sensor-based devices work (Devine et al., 2019). (2) In contrast to the existing devices such as Arduino[5] and Raspberry Pi,[6] the micro:bit is specifically created for educational purposes. The micro:bit is intentionally designed to have fewer difficulties for novice programmers to learn and construct embedded systems (e.g., programmers do not require to run a full operating system) (Devine et al., 2019).

2 https://makecode.microbit.org/.
3 https://python.microbit.org/.
4 https://scratch.mit.edu.
5 https://www.arduino.cc.
6 https://www.raspberrypi.org.

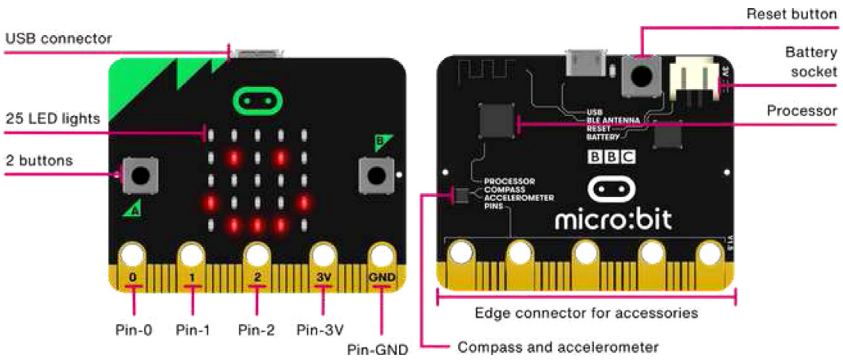

Fig. 1. The front and back of the micro:bit device.
Source: Taken from Micro:bit Educational Foundation (2021).

3. Methodology

The OzGirlsCT program was the first step of the "Women in STEM and Entrepreneurship" (WISE) program. The WISE program was a three-step education program with the following objectives: (1) understanding how girls develop and perceive CT; (2) developing a technology-focused entrepreneurial intention amongst girls (Shahin et al., 2021); and (3) increasing awareness of girls in STEM. The WISE program participants were Year 10 secondary school girls in the state of Victoria in Australia in 2019. Their ages ranged from 14 to 16. We refer to them as "girls" or "students" in this paper. This study only focuses on the OzGirlsCT program. In Australia, secondary schools last six years, and students should attend until age 17. Year 10 is the beginning of senior secondary school.

3.1. OzGirlsCT program

The OzGirlsCT program involved three one-day workshops. We organised these workshops in three days in 2019. In total, 203 girls from 44 secondary schools, grouped into 52 teams of 3–4 students, participated in three workshops (i.e., each team attended only one workshop). Each team was mentored by a "big sister" university mentor (hereafter "mentor").

3.1.1. OzGirlsCT program recruitment

3.1.1.1 Secondary school girls Information about the free OzGirlsCT program was sent to Victorian schools, and the schools advertised the OzGirlsCT program to their students through their internal communication channels. The schools used a range of methods to recruit participants within their schools. In some schools, teachers nominated students, while in others, students were able to self-nominate to the OzGirlsCT program. The students in each team were from the same school, and the formation of the teams was finalised by the schools. The friendship level of the students in the teams varied considerably, from those who did not know each other at all, to some of the team members in a team having close friendships. Initially, each school was able to send just one team; however, as we had additional capacity after the initial recruitment, schools that had requested participation from more than one team were able to send a second team.

3.1.1.2 Mentors Specifically, we sought women students at Monash University to recruit mentors for the OzGirlsCT program. We first designed an online Expression of Interest (EOI) form and advertised the EOI form through Monash University's newsletter, as well as sent it to the mailing list of IT women students at Monash University. In the EOI preamble, we described the purpose of our study and the characteristics we were looking for in potential mentors. Specifically, we sought women students who obtained or were doing a STEM-related degree at Monash University. They also had to have a basic level of programming skills. 66 women students completed the EOI. The next step involved interviewing the 66 candidates. 14 out of the 66 candidates did not attend the interviews. During the interviews, we asked the interviewees to describe the qualities and characteristics that make a good mentor for young girls. Then, we sought demographic information about the interviewees, including programming experience and mentoring, tutoring, or volunteering experience. We selected 31 women students as mentors. The main criteria used to select the 31 mentors were communication skills, personal characteristics (e.g., friendly and confident), tutoring and volunteering experience, and programming experience. Subsequently, we arranged a one-day training workshop to introduce the mentors to their responsibilities during the OzGirlsCT program and describe the OzGirlsCT program's objectives. Furthermore, mentors were trained on how to work and program with the micro:bit. Each mentor guided only one team per OzGirlsCT workshop. However, a few mentors guided teams across multiple workshops.

3.1.2 Pre-OzGirlsCT program activities

The only pre-requisite for entry into the OzGirlsCT program was the student's year level at school. We did not expect the students to have prior programming experience. A month before OzGirlsCT program workshops started, we provided introductory pre-work for the micro:bit to attempt to bring all students to the same level. More specifically, each student was given a micro:bit and encouraged to read the eBook created by our research team and do the exercises in the eBook. They were given resource links so that they could explore the micro:bit website (Micro:bit Educational Foundation, 2021) and a range of other tutorials and websites to learn about the micro:bit device features and how to code with the micro:bit. The eBook included several sample projects, tutorial videos, and exercises. Further, the student teams were also introduced to their mentors at this time and were encouraged to communicate with their mentors and ask any questions (e.g., debugging) regarding programming with the micro:bit, as they completed the pre-work. While we did not formally check that the activities were completed, students were told that they must complete all the required activities before their workshop, and were supported by their mentors to do so.

3.1.3 OzGirlsCT program activities

As described in Section 3.1.2, each student was given a micro:bit before the OzGirlsCT program started, and each student brought their micro:bit to the workshop. Students were also requested to bring their own laptops. Each workshop started

with familiarising students with the concepts of IoT. Next, students were asked to brainstorm the problems that they had in and/or observed around their everyday life. Following the suggestion proposed by Tissenbaum et al. (2019), students were not given a predefined problem or task (e.g., design a game based on predefined specifications), but were encouraged to develop computational ideas and solutions around realistic problems that were important to them and their communities. The members of each team then worked together to decide on which problem to focus on. While we encouraged team members to collaborate with each other, teams were responsible for choosing their working style and collaboration strategy as we followed problem-based learning. The difficulty levels of the problems chosen by the girls widely varied. For example, one team decided to improve the safety of cyclists at night, and another team was concerned about distracted students in class. The next step included completing the following problem statement for the identified problem: "*We believe that [the identified problem] … is a problem for [who] … because [reason] …*". Then, teams had to propose feasible solutions to solve their identified problem. The proposed solutions needed to have an IoT component. After choosing a solution to the identified problem, teams were requested to complete the statement: "*We could solve this problem by [solution idea] … and could demonstrate this on the micro:bit by [prototype idea] …*". In the next step, teams utilised the micro:bit device to prototype the proposed ideas. We encouraged the girls to verify their ideas, statements, and solutions in all steps by seeking feedback from other teams. Finally, each team had to present a business pitch discussing their products.

3.2 Data collection

To answer the research questions introduced in the Introduction section, we conducted a mixed-methods empirical study with a *concurrent triangulation strategy*, characterised by employing different data collection methods concurrently to confirm, cross-validate, and augment findings (Shull et al., 2007). We collected data from the participants of the OzGirlsCT program using three surveys and observations. Fig. 2 shows an overview of our research method. In the first step, a survey protocol was developed from the literature, grey literature, and the practical programming experience of the research team to collect data from students. Unlike formal literature, such as journal articles, grey literature refers to a body of materials such as government reports that have not been published and/or controlled by commercial publishers (Schöpfel and Farace, 2009). In addition, a survey protocol and an observation protocol were developed to collect data from mentors. In the next step, we ran two surveys to collect the students' perspectives on CT from different perspectives. Mentors were also asked to complete a survey and submit their observation reports.

3.2.1 Pre- and post-workshop student surveys

Protocol: We designed two surveys to collect data around the background, challenges, behavioural patterns, practices, and experiences of girls during the OzGirlsCT program. We carried out the first survey (i.e., pre-workshop survey) at the beginning of the OzGirlsCT workshops. In total, the pre-workshop survey consisted of 33 questions. In this paper, we only used three questions of the pre-workshop survey, which are demographic questions around prior coding experience, type of school, and class performance (e.g., "*Do you have any prior computer coding-related experience? If so, how long?*"). The rest of the pre-workshop survey questions sought the students' skills and interest in entrepreneurship and STEM. At the end of each OzGirlsCT workshop, students were asked to fill out a post-workshop survey with 40 questions. All questions, except one, were mandatory. Following the high-level objectives of the WISE program, these 40 questions focused on entrepreneurship (e.g., entrepreneurial inspiration), STEM, and CT. The following list details the questions of the post-workshop survey which were used in this study:

- **CT practices**. This study focused on a set of CT practices, including *planning*, *decomposition*, *abstraction*, *generalisation*, *algorithm*, *testing and debugging*, and *collaboration*. These practices are more likely to emerge and be evaluated when CT is introduced through programming activities (Denner et al., 2014; Shute et al., 2017; Kazimoglu et al., 2012). We designed 12 items to collect students' views on these CT practices, using age-appropriate terminology to indirectly measure these practices. Table 1 presents these CT practices, their respective items, and sources. We asked students to rate the difficulty of the 12 items for them (i.e., from *very difficult* to *very easy*).
- **Challenges and practices**. Through two mandatory open-ended questions, we asked students to share any challenges that they faced when implementing their ideas with the micro:bit and the practices or techniques that they used to overcome these challenges.
- **General comments**. Through an optional open-ended question, we let students share any general comments about the OzGirlsCT program.

Participants: The participants in the pre- and post-workshop surveys were the same Year 10 girls who attended the OzGirlsCT program workshops. Overall, 203 students participated in the workshops. Students came from different types of schools — public, private and Catholic, and most had good academic records as judged by their self-reporting. Whilst completing the surveys was voluntary, 197 out of 203 students completed each survey. We removed 4 responses from each survey because of the invalid nature of the responses (e.g., when we were not able to match a student's responses in the post-workshop survey to the pre-workshop survey due to Survey ID entry errors) (Meade and Craig, 2012). In the end, we received 193 valid responses for each survey (See Fig. 2).

3.2.2 Mentor survey

Protocol: The data collected from the post-workshop survey is based on students' self-reporting and self-assessment, which might be unreliable (Schwarz and Oyserman, 2001). To alleviate this limitation, we deployed an online survey with a similar goal of that of the post-workshop student survey to collect mentors' perspectives on the challenges, practices, and experiences of students during the OzGirlsCT program. For this study, mentors were asked to indicate the difficulty level of 12 CT practices for the team(s) that they mentored. These 12 CT practices were exactly the ones that were asked in the post-workshop student survey (See Table 1). Next, mentors shared the challenges that students faced when developing and implementing their ideas with the micro:bit and the practices used by students to address those challenges. We closed the mentor survey by asking mentors to share any comments and feedback they may have about the activities conducted during the OzGirlsCT workshops.

Participants: The participants in the mentor survey were the mentors recruited for the OzGirlsCT program (See Section 3.1.1.2). Mentors were asked to respond to the mentor survey at the end of the final OzGirlsCT workshop they participated in, as they may have participated in multiple workshops. We received 31 valid responses from the mentors.

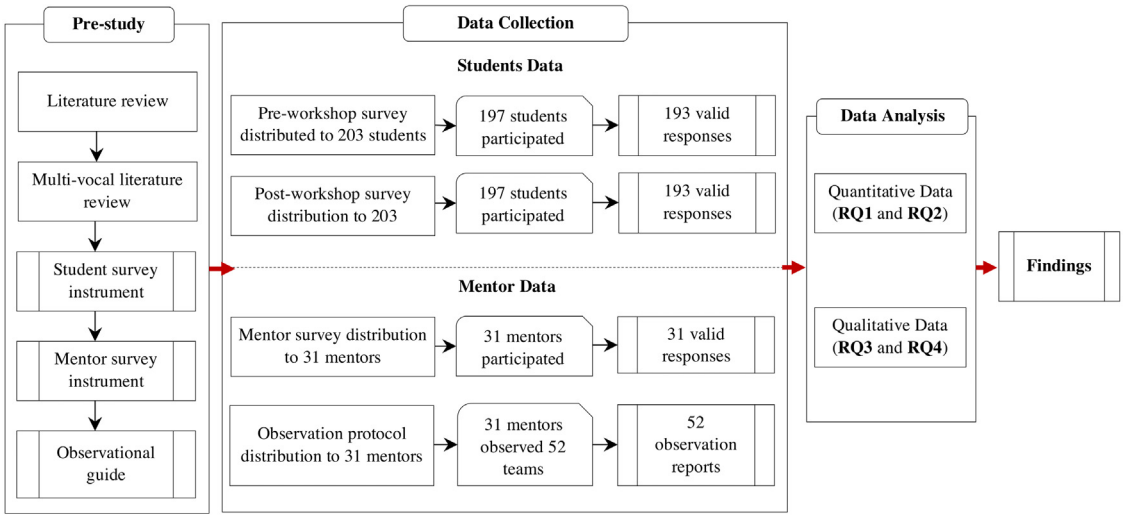

Fig. 2. Research method overview.

Table 1
CT practices, their respective items (statements), and sources.

CT practices	Item(s)	Sources
Planning	CTP1. Planning an idea before implementing it with the micro:bit	Kong (2019) and Zhong et al. (2016)
Decomposition	CTP2. Breaking down an initial idea into smaller, more manageable steps/parts	Atmatzidou and Demetriadis (2016)
Abstraction	CTP3. Leaving out the irrelevant detail/information in the description of an idea	Atmatzidou and Demetriadis (2016)
Generalisation	CTP4. Developing a general solution that can be applied to other problems in the future	Atmatzidou and Demetriadis (2016)
Algorithm	CTP5. Creating a series of ordered steps to implement an idea with the micro:bit CTP6. Exploring diverse solutions to an idea, until the ideal solution is achieved	Atmatzidou and Demetriadis (2016)
Testing and debugging	CTP7. Testing code frequently to check if it works CTP8. Identifying errors in code CTP9. Finding a solution to fix the identified errors in code	Lye and Koh (2014) and Kong (2019)
Collaboration	CTP10. Giving feedback to teammates and making suggestions to improve idea/code CTP11. Working collaboratively with team members CTP12. Reaching a consensus in group decisions	Grover et al. (2015), Berl and Lee (2011) and Astrachan et al. (2011)

3.2.3 Mentor observations

Protocol: Although a great deal of data can be gathered through surveys, the data gathered from surveys may be subjective and include potential biases such as social desirability (Malhotra and Birks, 2007). Hence, we used observation to collect actual and firsthand accounts about the work habits, behaviours, and interactions of students when developing and implementing their ideas with the micro:bit. Moreover, we wanted to identify the challenges and practices which students were unaware of or unable to communicate through the pre- and post-workshop surveys (Malhotra and Birks, 2007). Considering the numbers — 203 students grouped into 52 teams, it was not possible for us, as the authors of this paper (the researchers), to conduct a participant-observation study (Easterbrook et al., 2008). Therefore, we asked mentors to act as *observers* as well. During the one-day training workshop for mentors (discussed in Section 3.1.1.2), we instructed mentors about observation techniques. From the data collection perspective, it was a semi-structured observation (Malhotra and Birks, 2007). Whilst we provided an observation protocol to guide mentors on what to observe and collect, they were free to collect any data that they perceived as important. It is worth noting that mentors were not directly involved in their team's problem-solving process — their role was to guide and facilitate. The observation was accomplished with the think-aloud technique as students were asked to think out loud (i.e., verbalise their thought process) while working on their ideas (Shull et al., 2007). Since students may have sometimes forgotten to verbalise, mentors, as the observers, reminded them occasionally (e.g., every 15 min) to continue thinking out loud. The observation happened at the team level. Mentors submitted their team observation at the end of each OzGirlsCT workshop. It should be noted that as some mentors guided more than one team (each team attending a different OzGirlsCT workshop), the number of observation reports correlated to the number of teams. In the end, we collected 52 observation reports from mentors. The following questions were central to the observation protocol:

- Describe your observations on the team's motivations, thoughts, and assumptions.
- Describe your observations on how the members of the team interacted, communicated, and collaborated.
- Describe your observations on the team's work habits.
- Describe your observations on the issues that the team faced.
- Describe your observations on how the team fixed the issues.

Besides the above questions, the observation protocol included three single-choice questions. We asked mentors to indicate which of the following statements best applied to the workstyle of the team they mentored:

- All students were contributing evenly.
- A dominant student was guiding work, other students contributing.
- A dominant student was influencing work contributions, other students contributing unevenly.
- A dominant student was determining team effort, but some contribution from other students.

- Low student contribution, relying on one 'leader' to carry work.

The mentors were also asked to select one of the statements below to show the communication patterns within their mentored teams in two timeslots. The first timeslot was *ideation* time when students worked on developing their ideas. The second timeslot included *coding* time, in which students implemented their ideas with the micro:bit.

- All students were communicating freely: critical evaluations, objections, critiques, and opinions were freely exchanged within the team.
- All students were communicating well: some critical evaluations, objections, critiques, and opinions were exchanged within the team.
- All students were communicating intermittently: critical evaluations, objections, critiques, and opinions were exchanged but only sporadically.
- All students were communicating inadequately: critical evaluations, objections, critiques, and opinions were voiced but only with prompting from the mentor.
- Little to no communication, ideas, and critiques were rationed by a dominant student.

3.3 Data analysis

3.3.1 Quantitative analysis for RQ1 and RQ2

The close-ended questions, including the Likert scale and single-choice questions, were analysed using IBM SPSS Statistics 26 software to answer **RQ1** and **RQ2**. More specifically, we used the following statistical techniques: (i) We identified the most difficult CT practices for students (**RQ1**) by using the Scott–Knott Effect Size Difference (ESD) test proposed by Tantithamthavorn et al. (2017). We applied the Scott–Knott ESD test on the Likert scores of 12 CT practices from students' and mentors' perspectives. The main advantage of the Scott–Knott ESD test over the Scott–Knott test (Scott and Knott, 1974) is that it does not need normally distributed data. (ii) We applied the Mann–Whitney U test (Howell, 2012) to compare perceiving CT practices between students and mentors (**RQ1**). The Mann–Whitney U test was used because none of CT practices' scores was normally distributed (i.e., the Shapiro–Wilk test's p-values were less than 0.05 for all CT practices McCrum-Gardner, 2008). Also, the variables (i.e., CT practices) were measured at the ordinal level. Finally, a non-parametric Levene's test confirmed the equality of variances in both samples (i.e., p-values > 0.05 for all CT practices) (Nordstokke and Zumbo, 2010; Nordstokke et al., 2011). (iii) We conducted the Kruskal–Wallis one-way ANOVA tests (Howell, 2012) to understand the relationship between the difficulty level of CT practices and students' coding experience (**RQ2**). The homogeneity of variances in the studied samples was verified by the non-parametric Levene's test (i.e., p-values > 0.05 for all CT practices) (Nordstokke and Zumbo, 2010; Nordstokke et al., 2011). Furthermore, the pairwise post hoc tests (i.e., pairwise comparisons) (Howell, 2012; Field, 2013) were carried out on each pair of groups.

3.3.2 Qualitative analysis for RQ 3 and RQ4

We analysed the answers to the open-ended questions using open coding and constant comparison as the two main qualitative data analysis techniques of Grounded Theory (GT). The collected qualitative data was used to answer **RQ3** and **RQ4**. The qualitative analysis was supported by the NVivo software.[7] (Glaser and Strauss, 2017; Hoda and Noble, 2017) GT enables the researcher to identify the main concerns of the participants and understand how they address the concerns. This is achieved by constantly comparing data while the levels of abstraction are increasing (Glaser and Strauss, 2017; Hoda and Noble, 2017).

We first created three top-level nodes in NVivo according to our data sources (See Fig. 3): (1) student survey data, (2) mentor survey data, and (3) observation data. Since **RQ3** was about challenges and **RQ4** focused on practices, each high-level node (e.g., mentor survey data) was further decomposed into two sub-nodes: *challenge* node and *practice* node. Subsequently, the first author extracted the statements (e.g., **Statement_2** in Fig. 3) about the challenges that students faced and the statements (e.g., **Statement_1** and **Statement_3** in Fig. 3) about the best practices and techniques adopted by students in each of the data sources. Note that some statements (e.g., **Statement_1**) included both challenge(s) and the practice(s) used to address the challenge(s). Hence, we placed such statements in the *challenge* node and *practice* node to maintain the relationship between challenges and practices. In the next step, he performed open coding over multiple iterations to thoroughly analyse the data gathered from each data source. This step resulted in capturing key points in our data sources and assigning a label (i.e., code) to each key point. Fig. 3 shows the process of applying open coding on **Statement_1** and **Statement_3** identified two codes for the *practice* node. The analysis of **Statement_1** and **Statement_2** led to adding two codes to the *challenge* node (See Fig. 3).

The next step included performing the constant comparison technique to compare all codes identified in a data source against each other as well as to compare them with the codes from other data sources (Hoda, 2011). The identified codes from the previous step were iteratively grouped to generate *concepts*, and then the generated *concepts* were used to create *categories* (Hoda and Noble, 2017). In the next step, the identified *codes*, *concepts*, and *categories* were shared with the second author for review. Then, the first author and the second author held several face-to-face meetings to discuss the *codes*, *concepts*, and *categories* and solve any disagreements and inconsistencies and reach a consensus on the final list of *codes*, *concepts*, and *categories*. Fig. 4 shows how performing the constant comparison technique on four concepts produced the category, "*Establishing a collaborative and supportive culture within the team*".

4 Results

This section presents the findings of the RQs. It should be noted that when we refer to data from the student survey, we use **SSXY** notation. In our notation, **X** (1…52) shows the team number of a student, and **Y** (1…4) refers to the student number in a team. For instance, **SS153** refers to *student 3* within *team 15* from the student survey. The participants (mentors) in the mentor survey and observation reports are presented as **MSX** and **MOY**, respectively, in which **X** (1…31) is the participant number and **Y** (1…52) indicates the team number. For instance, an excerpt from the observation report of *team 37* is marked as **MO37**.

4.1 Secondary school girls demographic data

Fig. 5 shows an overview of students' demographics. 97 out of 193 (50.2%) students had at least one month of coding experience, 32 students (16.5%) had less than 1-month of coding experience, while 69 students (35.7%) had no coding experience before the OzGirlsCT program. 84 (43.5%) students came from Government schools. The rest from Catholic schools (54 girls, 27.9%) and Independent schools (55 girls, 28.4%). Almost 97% of the girls designated that their academic performance in the class ranged from 70%–79.99% to 90%–100%.

[7] http://www.qsrinternational.com.

Statement_1 from Student Survey

Raw data: *"It was originally very hard to incorporate the usage of the micro: bit in our original design, however, after we changed our design a few times and came up with different ideas, we were able to use the micro: bit effectively".*

Key point: *"Hard to the original design into our original design", "Changing the original design to the new ones and select the feasible one for micro:bit"*

Code: Incorporating idea into the micro:bit, Generating as many as possible ideas/solutions

Statement_2 from Mentor Survey

Raw data: *"Too scared to explore functions and do things on their own".*

Key point: *"Being scared"*

Code: Fear

Statement_3 from Mentor Observation

Raw data: *"They tried to limit scope creep by forcing themselves to focus on the most important functionalities and then listed the extra functionalities for next steps…. ".*

Key point: *"Focusing on the most important functionalities", "Extra functionalities for next steps"*

Code: Incremental approach

Fig. 3. Examples of constructing codes.

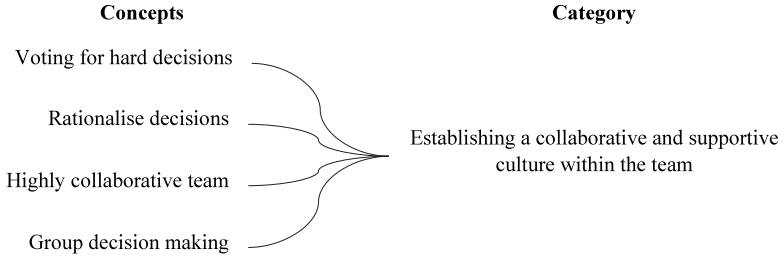

Fig. 4. Building a category by applying constant comparison.

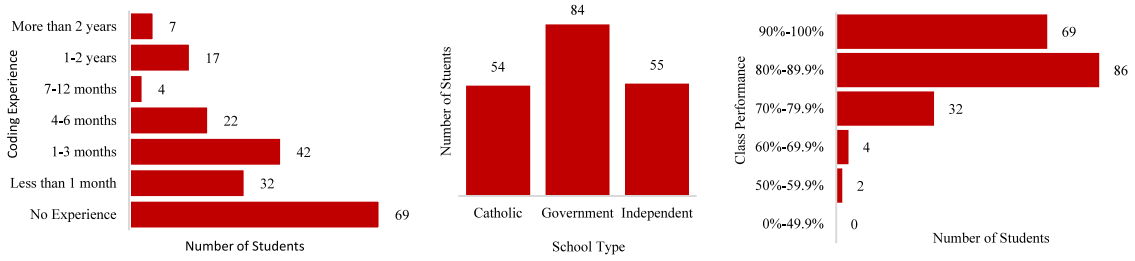

Fig. 5. Secondary school girls' demographic data.

4.2 What are the perceptions of secondary school girls on the difficulty level of CT practices when doing collaborative programming with the micro:bit? (RQ1)

To understand our participants' perspectives on CT practices, we provided 12 statements (See Table 1) for which students could indicate the difficulty level of CT practices and mentors could indicate the difficulty level of CT practices for the team(s) that they mentored. Another goal of these statements is to understand whether the perceptions of CT practices differ based on the participant role (students vs. mentors). To identify the most difficult CT practices for students, we performed the Scott–Knott Effect Size Difference (ESD) test (Tantithamthavorn et al., 2017) on the Likert scores of 12 CT practices from students' and mentors' perspectives. Table 2 shows that CT practices are classified into 6 statistically different groups in terms of difficulty for all respondents. In Table 2, a group with a smaller number has more difficult CT practices. We observe that CTP9, CTP8, and CTP3 are among the most difficult practices (Group 1) for students to apply. Interestingly, 2 of these CT practices (CTP9 and CTP8) are related to testing and debugging. Table 3 presents the mean and median Likert scores of 12 CT practices. As shown in Table 3, students' average Likert scores of CTP8 ("*identifying errors in code*") and CTP9 ("*finding a solution to fix the identified errors in code*") are 3.18 and 3.15 respectively, while mentors scored CTP8 (average Likert score: 2.77) and CTP9 (average Likert score: 2.87) lower. We also find that CTP11 ("*working collaboratively with team members*"), CTP12 ("*reaching a consensus in group decisions*"), and CTP10 ("*giving feedback to teammates and making suggestions to improve idea/code*") are among the top 3 easiest CT practices to apply.

Fig. 6 shows how students and mentors rated the difficulty level of each of the CT practices. As a student, the majority of the respondents claimed that "*working collaboratively with team members*" (i.e., 87% rated CTP11 as *easy* or *very easy*) and "*reaching a consensus in group decisions*" (i.e., 80% rated CTP12 as *easy* or *very easy*) are the easiest CT practices to apply. Mentors had the same feeling, as at least 87% of mentors believed that applying CTP11 and CTP12 is an easy or very easy task for students. On the other hand, the following were rated by students as the most difficult CT practices to apply: (1) finding a solution to fix the identified errors in code (CTP9: 29% *difficult* or *very difficult*, 34% *neutral*, 37% *easy* or *very easy*); (2) identifying errors in code (CTP8: 28% *difficult* or *very difficult*, 34% *neutral*, 38% *easy* or *very easy*); and (3) leaving out the irrelevant detail/information in the description of an idea (CTP3: 24% *difficult* or *very difficult*, 38% *neutral*, 37% *easy* or *very easy*). Mentors had slightly different observations on the most challenging CT practices for students. They ranked CTP8 (42% *difficult* or *very difficult*, 23% *neutral*) as the most difficult CT practice for students to implement, followed by CTP1 (35% *difficult* or *very difficult*, 19% *neutral*) and CTP9 (32% *difficult* or *very difficult*, 26% *neutral*). In contrast to mentors, it is a commonly held belief among students that "*planning an idea before implementing it with the micro:bit*" (CTP1) is a relatively easy task, as only 16% of them rated this CT practice as *difficult* or *very difficult*. In addition, CPT6, CTP7, CPT10, CTP11, and CTP12 are the only CT practices that over 50% of students and mentors mutually believed are (very) easy practices to apply. Interestingly,

Table 2
CT practices are grouped into six statistically distinct groups in terms of difficulty (Scott–Knott Effect Size Difference test applied to all respondents) — the smaller the group number, the more difficulty the CT practices.

Group	CT practices
1	**CTP9**: Finding a solution to fix the identified errors in code **CTP8**: Identifying errors in code **CTP3**: Leaving out the irrelevant detail/information in the description of an idea
2	**CTP5**: Creating a series of ordered steps to implement an idea with the micro:bit **CTP6**: Exploring diverse solutions to an idea, until the ideal solution is achieved **CTP4**: Developing a general solution that can be applied to other problems in the future **CTP1**: Planning an idea before implementing it with the micro:bit **CTP2**: Breaking down an initial idea into smaller, more manageable steps/parts
3	**CTP7**: Testing code frequently to check if it works
4	**CTP10**: Giving feedback to teammates and making suggestions to improve idea/code
5	**CTP12**: Reaching a consensus in group decisions
6	**CTP11**: Working collaboratively with team members

these CT practices, except one (CTP7), can be classified as soft skills (Debnath et al., 2012).

Not all practices were ranked as *very difficult*. Regarding which practices the respondents did not perceive as *very difficult* during ideation and coding, we have "*breaking down an initial idea into smaller, more manageable steps/parts*" (14% of students and 26% of mentors rated CPT2 as *difficult*) and "*reaching a consensus in group decisions*" (6% of students and 3% of mentors rated CPT12 as *difficult*). Furthermore, CPT10 ("*giving feedback to teammates and making suggestions to improve idea/code*") and CTP5 ("*creating a series of ordered steps to implement an idea with the micro:bit*") were never ranked as *very difficult* by students and mentors respectively.

The fifth column in Table 3 presents the mean difference in the average Likert scores of students and mentors. We were interested in understanding whether there is a significant difference in perceiving CT practices between students and mentors. Table 3 shows the results of the Mann–Whitney U test. We observe that there is a significant difference only for CTP2 ($U = 2227.0$, $N1 = 193$, $N2 = 31$, $p\text{-value} = 0.014$, $r = 0.164$) (Cohen, 2008), indicating that the difficulty level of "*breaking down an initial idea into smaller, more manageable steps/parts*" was perceived differently by mentors (median = 3, mean rank = 93.73) than by students (median = 4, mean rank = 115.52), with mentors seeing it as being significantly more difficult.

4.3 Is there a relationship between secondary school girls' prior coding experience and their perceptions of the difficulty of CT practices? (RQ2)

As previously noted, we collected the level of students' coding experience in the pre-workshop survey. To understand the effect of students' coding experience in their perceptions of CT practices, we first divided students' data into three independent groups. The first group included 69 students who had no coding experience before the OzGirlsCT workshops (See Table 4). The second group consisted of 96 students who had less than 6-month coding experience (i.e., relatively low experienced group). The third one included the students who had relatively moderate experience in coding, indicating more than 6-month experience (i.e., 28 students). Then, the Kruskal–Wallis one-way ANOVA tests (Howell, 2012) were conducted on 3 independent variables (*no experience* group, *relatively low experience* group, and *relatively moderate experience* group) and 12 dependent variables (CTP1…CTP12). The results of the Kruskal–Wallis one-way ANOVA tests are summarised in Table 4.

We observe that the level of coding experience significantly affects how students perceived CTP8 and CTP9. The students who indicated higher coding experience had lower difficulty in "*identifying errors in code*" (CTP8, $\chi^2(2) = 9.987$, $p\text{-value} = 0.007$) and "*finding a solution to fix the identified errors in code*" (CTP9, $\chi^2(2) = 7.790$, $p\text{-value} = 0.020$). The results of the pairwise post hoc tests (i.e., pairwise comparisons) are shown in Tables 5 and 6, in which each row tests the null hypothesis that the distributions of each pair group are equal. Concerning the perception of CTP8, we observe that a statistically significant difference exists between the group who had no experience in coding and those who had moderate coding experience (adjusted $p\text{-value} = 0.0012$, adjusted using the Bonferroni correction). As shown in Table 5, there are no differences between the *no experience* group and *relatively low experience* one (adjusted $p\text{-value} = 0.055$) or *relatively low experience* group and *relatively moderate experience one* (adjusted $p\text{-value} = 0.603$). We observe the same pattern for CTP9 (See Table 6). The perception of CTP9 differs significantly from the students without any coding experience to the moderately experienced students in coding (adjusted $p\text{-value} = 0.041$).

4.4 What challenges do secondary school girls face when collaboratively implementing computational ideas with the micro:bit? (RQ3)

To understand the challenges and barriers that students experienced while developing and implementing computational ideas with the micro:bit, we asked students the following question: "*What issues and challenges did you face when implementing your ideas with the micro:bit?*" We also solicited mentors' perspectives and observations in this regard. We identified six categories of challenges. Fig. 7 visualises the identified challenges. This visualisation enables a reader (i.e., researcher or practitioner) to quickly get an overview of the challenges faced by girls when developing and implementing a computational idea with the micro:bit. In this figure, the outer layer influences all inner layers. For example, as described later in this section, some of the challenges related to "Incorporating idea into the micro:bit" or "Code debugging" can be attributed to the limitations of the micro:bit device. Furthermore, Fig. 7 shows that there might be dependencies among the challenges (e.g., increase) in a layer. For example, the complexity of code can increase the challenges related to code debugging. Below, we describe each of these six challenges. For brevity, we only include a few quotations from our participants. Table 7 sorts these challenges based on the frequency of their appearance in the data and shows three illustrative quotations corresponding to each of these six challenges. Table 7 also indicates the frequency of sub-challenges (also presented in Fig. 7) that appeared in each identified challenge. Note that the summation of the frequency of sub-challenges in a challenge is sometimes less than the total number of that challenge. For example, while the "incorporating idea in the micro:bit" challenge appeared in total 76 times in our data, we only found three reasons (sub-challenges) behind this challenge: "breaking down or narrowing an idea" (n = 18),

Table 3
CT practices and the mean and median Likert scores (very difficult = 1, difficult = 2, neutral = 3, easy = 4, very easy = 5).

ID	Computational thinking practices	Student (n = 193) Mean (Median)	Mentor (n = 31) Mean (Median)	Mean difference	*p-value*
CTP1	Planning an idea before implementing it with the micro:bit	3.54 (4)	3.16 (3)	0.38	0.068
CTP2	Breaking down an initial idea into smaller, more manageable steps/parts	3.57 (4)	3.16 (3)	0.41	**0.014***
CTP3	Leaving out the irrelevant detail/information in the description of an idea	3.19 (3)	3.10 (3)	0.09	0.624
CTP4	Developing a general solution that can be applied to other problems in the future	3.53 (4)	3.26 (4)	0.27	0.234
CTP5	Creating a series of ordered steps to implement an idea with the micro:bit	3.46 (3)	3.10 (3)	0.36	0.055
CTP6	Exploring diverse solutions to an idea, until the ideal solution is achieved	3.40 (4)	3.45 (4)	0.05	0.707
CTP7	Testing code frequently to check if it works	3.64 (4)	3.68 (4)	0.4	0.640
CTP8	Identifying errors in code	3.18 (3)	2.77 (3)	0.41	0.115
CTP9	Finding a solution to fix the identified errors in code	3.15 (3)	2.87 (3)	0.28	0.208
CTP10	Giving feedback to teammates and making suggestions to improve idea/code	3.80 (4)	3.90 (4)	0.1	0.236
CTP11	Working collaboratively with team members	4.34 (5)	4.48 (5)	0.14	0.144
CTP12	Reaching a consensus in group decisions	4.16 (4)	4.35 (4)	0.19	0.284

*Significant at $p < 0.05$ (Mann–Whitney U statistical test).

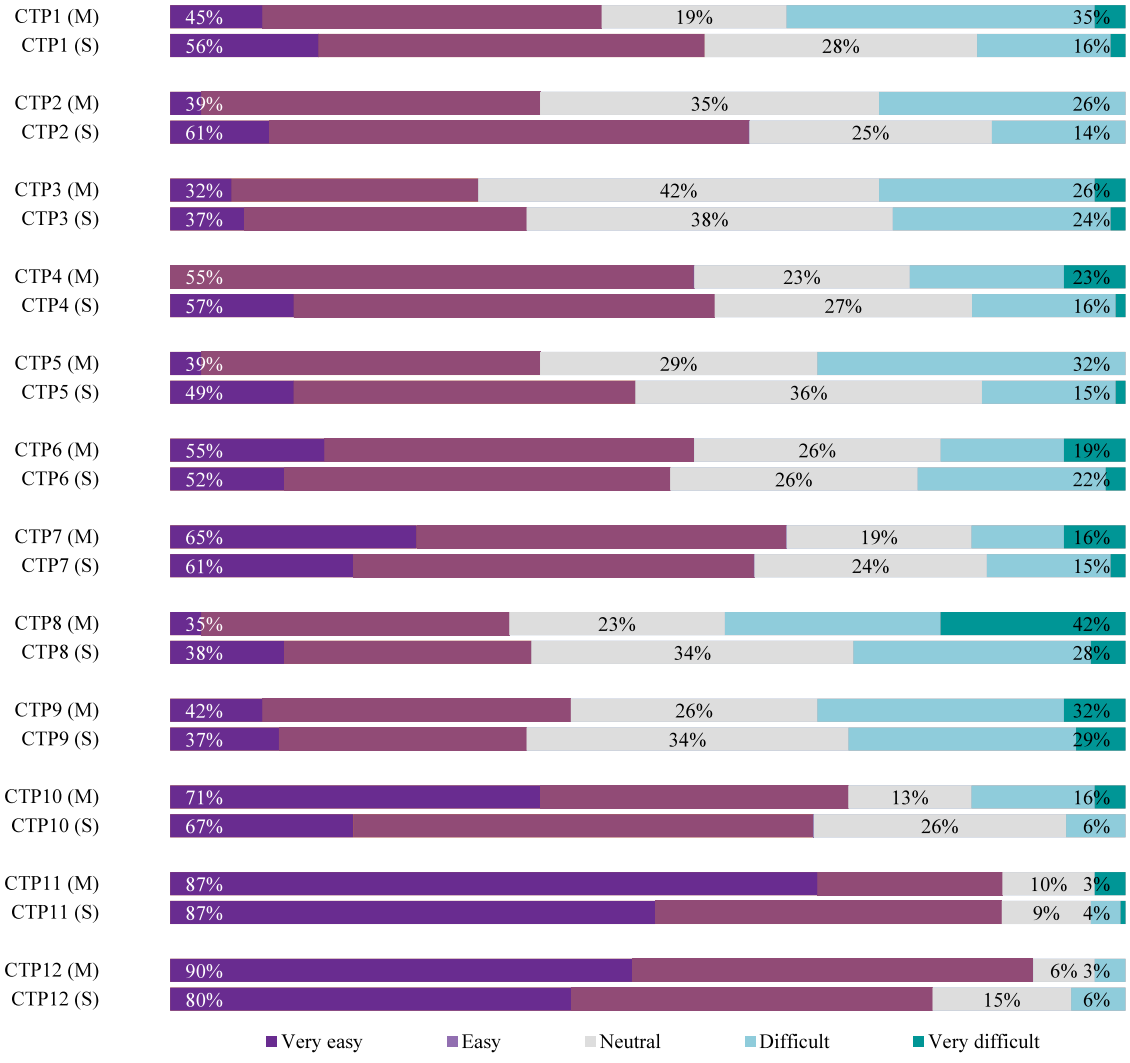

Fig. 6. Secondary school girls' (n = 193) and mentors' (n = 31) perceptions of CT practices (M = Mentor and S = Student).

"creating a series of ordered steps to implement an idea" (n = 17), and "leaving out irrelevant/unimportant stuff from an idea" (n = 8). For the rest of the references (N = 33) referring to this challenge, while the participants mentioned this challenge, they did not provide any reasons behind it.

Incorporating idea into the micro:bit. The most frequently reported challenge was transferring the planned idea to the micro:bit (e.g., *"It was a bit difficult to convert my worded problem into one which the code could follow"*. ***SS11***). Our analysis shows that this challenge mainly stems from the fine-grained challenges expressed in the open-ended questions, including the **difficulty** in "breaking down or narrowing an idea", "leaving out irrelevant/unimportant stuff from an idea", and "creating a series of ordered steps to implement an idea". In Section 4.2, the participants rated that it was *moderately* difficult to apply CTP2,

Finding 1. *Students and mentors do not differ significantly in perceiving the difficulty level of applying computational thinking practices.*

Finding 2. *The aggregated students' and mentors' opinions indicate that "debugging" (i.e., measured by the statements "finding a solution to fix the identified errors in code" and "identifying errors in code") is the most difficult computational thinking practice to apply, followed by "abstraction" (i.e., measured by the statement "leaving out the irrelevant detail/information in the description of an idea").*

Finding 3. *From the participants' perspective, collaborative practices of computational thinking, including "reaching a consensus in group decisions", "working collaboratively with team members", and "giving feedback to teammates and making suggestions to improve idea/code" are the easiest practices to apply.*

Table 4
Testing the effect of girls' coding experience in their perceptions of CT practices.

Variables	Level of coding experience			
	Mean (Median)			p-value
	No (n = 69)	Low (n = 96)	Moderate (n = 28)	
CTP1: Planning an idea before implementing it with the micro:bit	3.55 (4)	3.52 (4)	3.61 (4)	0.931
CTP2: Breaking down an initial idea into smaller, more manageable steps/parts	3.51 (4)	3.60 (4)	3.61 (4)	0.680
CTP3: Leaving out the irrelevant detail/information in the description of an idea	3.14 (3)	3.27 (3)	3.04 (3)	0.486
CTP4: Developing a general solution that can be applied to other problems in the future	3.45 (4)	3.65 (4)	3.32 (3.5)	0.198
CTP5: Creating a series of ordered steps to implement an idea with the micro:bit	3.33 (3)	3.46 (3)	3.75 (4)	0.206
CTP6: Exploring diverse solutions to an idea, until the ideal solution is achieved	3.29 (3)	3.53 (4)	3.25 (3.5)	0.228
CTP7: Testing code frequently to check if it works	3.48 (4)	3.68 (4)	3.93 (4)	0.083
CTP8: Identifying errors in code	2.88 (3)	3.27 (3)	3.57 (3)	**0.007***
CTP9: Finding a solution to fix the identified errors in code	2.86 (3)	3.25 (3)	3.50 (3)	**0.020***
CTP10: Giving feedback to teammates and making suggestions to improve idea/code	3.67 (4)	3.86 (4)	3.93 (4)	0.125
CTP11: Working collaboratively with team members	4.26 (4)	4.36 (5)	4.43 (5)	0.341
CTP12: Reaching a consensus in group decisions	4.00 (4)	4.25 (4)	4.25 (4)	0.103

*Significant at $p < 0.05$ (Kruskal–Wallis one-way ANOVA).

Table 5
Pairwise comparisons for CTP8 (grouped based on the level of coding experience).

	Test statistic	Standard error	Standardised test statistic	*p*-value	Adjusted *p*-value
No Exp. vs. Low Exp.	20.01	8.49	2.357	0.018	0.055
No Exp. vs. Moderate Exp.	34.786	12.053	2.886	0.004	**0.012***
Low Exp. vs. Moderate Exp.	−14.775	11.553	−1.279	0.201	0.603

*Significant at $p < 0.05$.

Table 6
Pairwise comparisons for CTP9 (grouped based on the level of coding experience).

	Test statistic	Standard error	Standardised test statistic	*p*-value	Adjusted *p*-value
No Exp. vs. Low Exp.	18.747	8.499	2.206	0.027	0.082
No Exp. vs. Moderate Exp.	29.727	12.065	2.464	0.014	**0.041***
Low Exp. vs. Moderate Exp.	−10.98	11.565	−0.949	0.342	1

*Significant at $p < 0.05$.

Finding 4. *Having prior experience in coding significantly reduces the difficulty level of "identifying errors in code" and "finding a solution to fix the identified errors in code".*

Finding 5. *We have no evidence to suggest that the level of coding experience affects the perceived difficulty level of any other computational thinking practices.*

CTP3, and CTP5, which are linked to these fine-grained challenges, respectively. We found frequent references in our data sources that students struggled to break down their ideas in a way that could be implemented with the micro:bit. Students also frequently complained about how and where to start coding (e.g., *"I didn't have enough coding knowledge to know where to start, so I struggled with such a challenge". **SS22***). These challenges seem to be magnified by a lack of coding experience. The simplicity of the micro:bit meant that students had to essentially leave out the unnecessary details and features from their ideas and solutions. **MO20** pointed out:

> *"The team initially wanted to jam-pack their idea with many features. However, I had to bring it down for them to decide which features they thought were the most important and to focus on a few key features that would really sell the idea." **MO20***

Code debugging. As we discussed in Section 4.2, our respondents rated the testing- and debugging-related practices (i.e., CTP7,

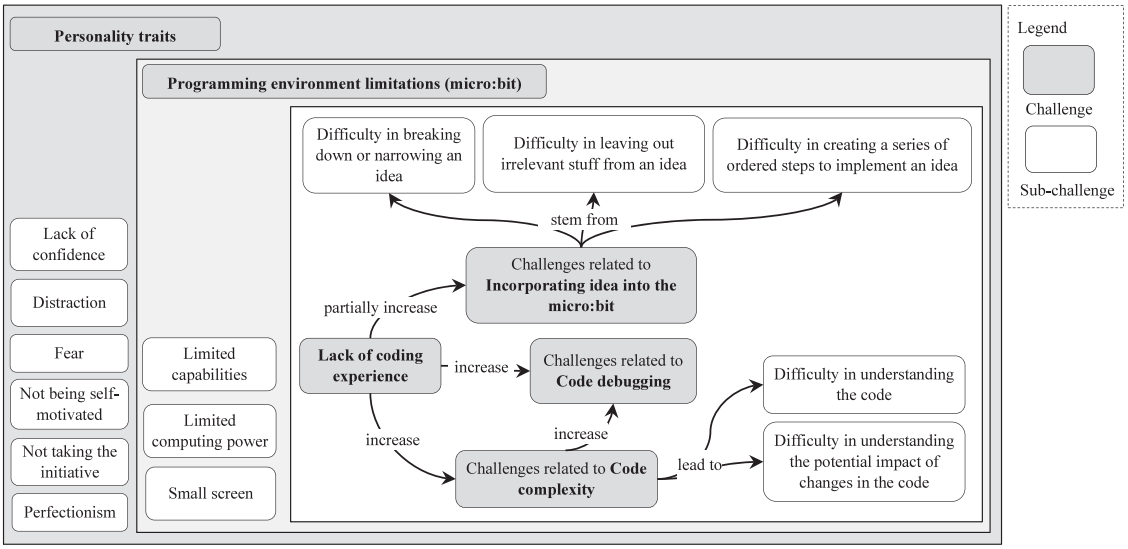

Fig. 7. An overview of challenges that secondary school girls faced when working on computational solutions and the relationship among them.

Table 7
Illustrative quotes for the challenges that secondary school girls faced when working on computational solutions (**N** shows the frequency of each challenge that appeared in total in the data).

Theme (challenge)	N	Illustrative quotes
Incorporating idea into the micro:bit *Breaking down or narrowing an idea* (**18**) *Creating a series of ordered steps to implement an idea* **(17)** *Leaving out irrelevant/unimportant stuff from an idea* **(8)**	**76**	"*Certain ideas we had could not be properly interpreted on the micro:bit.*" **SS63** *"As many of them [students] had a little coding experience, they often had no idea how to translate their ideas and concepts into smaller, manageable steps to be implemented into the code."* ***MS12*** *"Choosing what specific aspects [of the idea] to display on the micro:bit [was a challenge]."* ***SS12***
Code debugging	**41**	*"The code had a few confusing errors that I didn't know how to fix."* ***SS82*** *"They struggled with a small piece of code that seemed to malfunction. Being able to test what part of the code was going wrong was a barrier, but that could be rectified with some more practice in debugging code."* ***MS14*** *"They often stuck on the errors and got so frustrated when they believe they cannot solve the problem."* ***MS29***
Personality traits *Lack of confidence* **(7)** *Distraction* **(6)** *Fear* **(5)** *Not being self-motivated* **(3)** *Not taking the initiative* **(2)** *Perfectionism* **(2)**	**25**	*"The girls often got side-tracked and overexcited."* ***MO14*** *"At the end when the 2 of them were getting easily distracted, 1 of the other girls (the writer) would ask the girls to focus and pull their attention back".* ***MO7*** *"[Students] waited for me to do something, sat there, [and they were] quiet until I did something."* ***MO2***
Code complexity *Difficulty in understanding the code* **(9)** *Difficulty in understanding the potential impact of changes in the code* **(4)**	**22**	*"[The team members were] not testing small parts and then realising after a big piece of code was written that it didn't work as expected and getting frustrated."* ***MS15*** *"Even if I wanted to change things in the code (which required changing in the JavaScript form), it was difficult for me."* ***SS511*** *"We faced some problems with getting all of the code to work."* ***SS291***
Micro:bit limitations *Limited capabilities* **(14)** *Limited computing power* **(5)** *Small screen* **(3)**	**22**	*"It [micro:bit] doesn't have everything that would actually make what we wanted, however, we managed to make something similar to what we wanted to address the issues and challenges, the group consulted with each other and asked our mentors for help when we reached a stump in our thinking."* ***SS201*** *"We also had some issues with the micro bit as it was not as flexible as we would like it to be in terms of functionality."* ***SS224*** *"This was hard to implement on the micro:bit device as the device has a limited computing power."* ***MO9***
Coding experience	**21**	*"I think it was simply the lack of experience I had with coding (besides the pre-work and basic coding in Year 7) and my unfamiliarity with the functions of the coding blocks that made it quite difficult to implement the ideas with the micro:bit."* ***SS161*** *"There were still a few variables that I was not known of."* ***SS503*** *"They have limited education about programming; thus, they only could make the prototype of their product."* ***MS26***

CTP8, and CTP9) as the most difficult CT practices to apply. Through the open-ended questions in the surveys and observation study, many respondents attempted to provide clarifications and reasons for their answers to CTP7, CTP8, and CTP9 (e.g., *"It was hard to fix the errors since we didn't actually know what the error in the code was"* ***SS124)***. One mentor, **MS14**, pointed out these challenges could be mitigated by gaining more experience in code debugging. According to **MS8**, **MS15**, and **M15**, the inability to quickly resolve the issues with the code seems to generate frustration and demotivate the students. The students tried *not*

to make even simple mistakes in the code because they felt they could not solve the issues in the code. One student elaborated on:

> *"When we found flaws in our product we were creating, particularly ones that we took a long time (or didn't manage at all) to find solutions for, it kind of demotivated me for a little while."* ***SS163***

Personality traits. Besides the technical problems, the respondents reported a group of personality traits as challenges. This confirms the findings of Ketenci et al. (2019) and Román-González et al. (2018) that the learning outcomes of a learner in CT are related to the learner's characteristics (i.e., non-cognitive side of CT). Mentors observed that some students did not have complete confidence in themselves. This resulted in students who *could not* or *did not want* to explore the problem (i.e., ideation) and solution (i.e., coding) spaces properly. One mentor explained how a lack of confidence led to self-filtering in the ideation phase within a team:

> *"Students were quite unsure of themselves and tended to dismiss their ideas. While no idea was initially written down then discarded, I suspect that their ideation phase had a lot of self-filtering going on as students tended to stare intently at their post-it notes. There was also quite a lot of self-doubt, especially when working on their prototype coding."*

This lack of confidence can be exaggerated by fear of suggesting something wrong or being wrong (e.g., *"[My students were] too scared to explore functions and do things on their own"* ***MS2***). Distraction was another issue related to the personality traits, as some mentors described that it was challenging for students to remain focused. Fewer mentors also specified that students were not self-motivated and did not show initiative or they were exceptionally perfectionist (e.g., "*Some of the members were perfectionists, and I had to stop them from completely erasing their idea when they were struggling to find a solution"* ***MO22***).

Code complexity. The respondents frequently shared experiencing issues due to the complexity of code on understanding the code, as mentioned by **SS131**: "*We found it hard to make our solution [code] clear and uncomplicated"*. **SS311** shared that it was challenging to "*learn and remember what codes did what and in what order they had to go in"*. Other students struggled to figure out the potential impact of changes in the code, as mentioned by **SS51**: *"[It was difficult to] learn what code did what and learning if we added a code it affected everything"*.

Micro:bit limitations. The participants reported challenges regarding the micro:bit device frequently. Among these, the limited capabilities of the micro:bit was the prominent one, especially from the students' perspective. Here is one of the examples indicating students believed that the limited capabilities of the micro:bit did not allow them to completely implement their ideas:

> *"When implementing our ideas with the micro:bit, we struggled with its capabilities and spent a bit of time researching how to solve these limits. In the end, we made a very simplified version [of our idea], accepting that the micro:bit could not handle the final product."*

Some respondents described the computing challenges related to the micro:bit and reported that the micro:bit device had limited computing power to execute the complex code (e.g., "*The micro:bit does not have the capability to execute code with the level of complexity that my code possessed*" ***SS302***). Having a small screen and a limited number of functions were also reported as issues related to the micro:bit.

Coding experience. In some cases, the (technical) challenges that emerged from our data related to the students' experience and expertise in coding. Having adopted the experiential learning approach, some respondents reported that it was challenging to implement the ideas with the micro:bit and ensure the code worked properly with the limited coding experience and knowledge that students had. This can be vividly exemplified by the following quote:

> *"I think it was simply the lack of experience I had with coding (besides the pre-work and basic coding in Year 7) and my unfamiliarity with the functions of the coding blocks that made it quite difficult to implement the ideas with the micro:bit."* ***SS161***

4.5 What practices do secondary school girls employ to overcome these challenges? (RQ4)

The practices and techniques that students employed to overcome the challenges experienced during the ideation and coding were collected through a compulsory open-ended question in each of our data sources: the student survey, the mentor survey, and the observation protocol. Our qualitative analysis described in Section 3.3.2 revealed that students used six main practices during the ideation and coding using the micro:bit device. Fig. 8 shows these six practices and indicates which challenges presented in Section 4.4 are (partially) addressed by these practices. We further visualise the relationship among these practices in Fig. 9. In Fig. 9, the outer layer influences all inner layers. This section defines, elaborates, and provides examples of each practice. We describe the six identified practices with a few quotations from our participants. For brevity, we only include a few quotations from our participants. Table 8 sorts these practices based on the frequency of their appearance in the data and shows three illustrative quotations corresponding to each of these six practices.

Feedback-driven development. Our analysis shows that the most common practice used by students was the feedback-driven development practice (i.e., seeking help and feedback) (Beller, 2018), in which students constantly received feedback by consulting with their team members and mentors. Students mainly used the feedback-driven development practice to enhance the quality of their code and ideas. **SS201** commented on the importance of the received feedback during the ideation phase, as shown by the following:

> *"To address the issues and challenges, the group consulted with each other and asked our mentors for help when we reached a stump in our thinking." SS201*

Whilst students mainly provided feedback to and received feedback from their teammates and mentors, some students attempted to obtain authentic and diverse feedback from the members of other teams as well. Our observation shows that as the workshop day progressed, students were more willing to seek feedback and ask more questions. According to **MO31**, this was an effective practice as students became conscious of seeking help instead of backing out.

Establishing a collaborative and supportive culture within the team. Several participants discussed the effects of working as *a team* to address the challenges and issues. We found that a collaborative and supportive team implies that the team explores the issues collaboratively through scenarios and subsequently resolves the issues in that context whereby each team member evenly provides inputs on possible ideas and solutions. A mentor described the characteristics of the collaborative and supportive team as follows:

Finding 6. *According to our participants, incorporating an idea into the micro:bit device as a hybrid block/text programming environment is the most challenging task while developing and implementing a computational idea.*

Finding 7. *Participants provide evidence that the challenges and barriers faced when developing and implementing computational ideas can also be attributed to personality traits. This qualitative result is consistent with the existing literature (Ketenci et al., 2019; Román-González et al., 2018), which quantitatively reveals the existence of a non-cognitive side of computational thinking.*

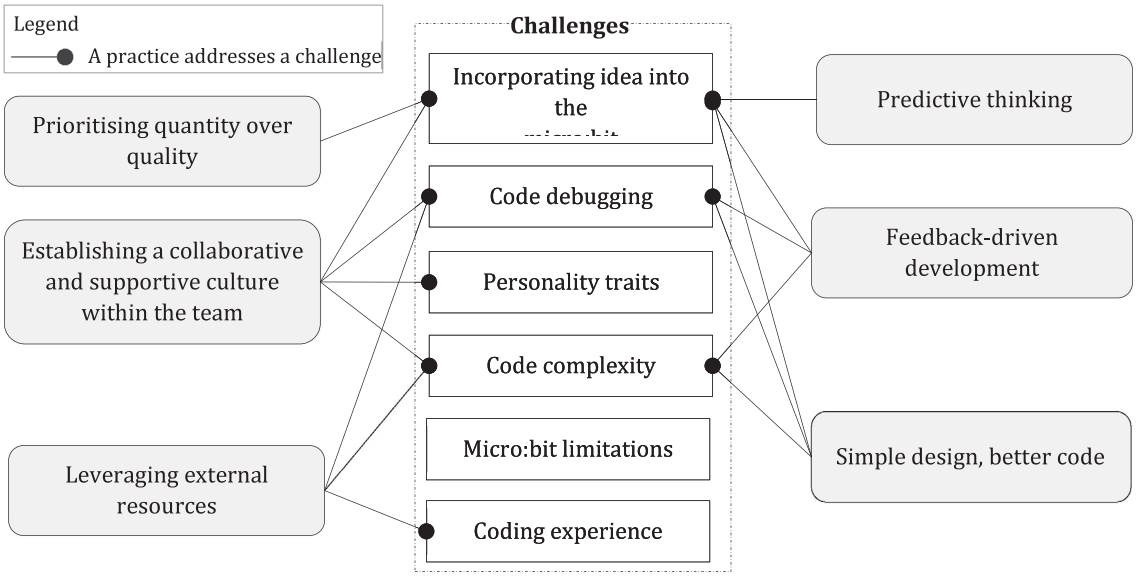

Fig. 8. The relationship between the identified challenges and practices.

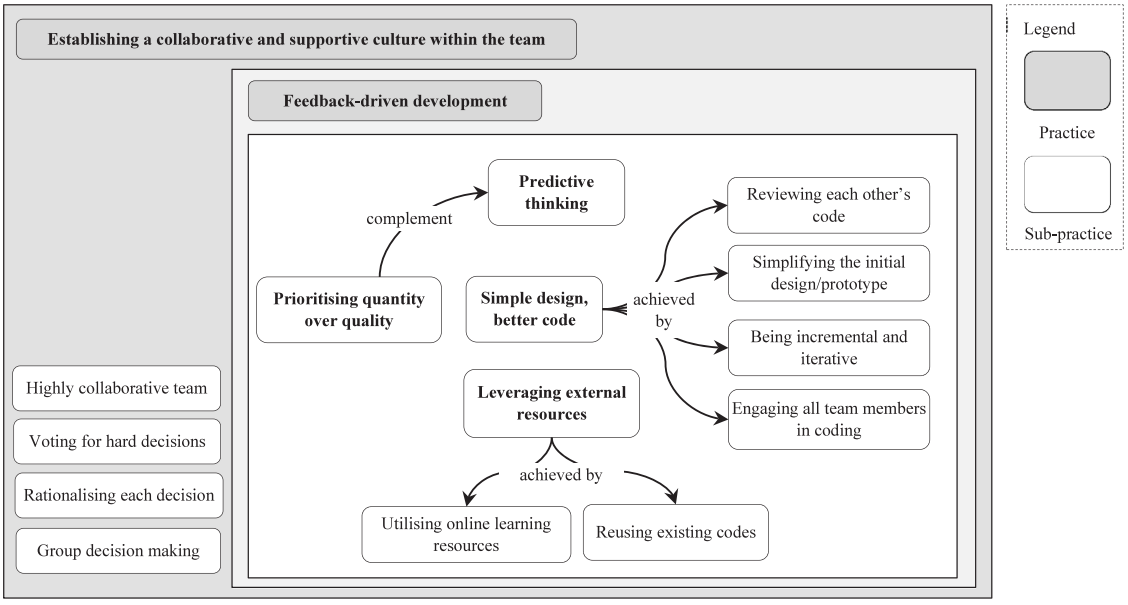

Fig. 9. An overview of practices employed by secondary school girls and the relationship among them.

"The issue that they encountered were discussed evenly across the members with each providing their own take on the issue that was considered equally by each member." ***MO10***

The quantitative analysis of three single-choice questions from the observation reports confirms that the majority of the teams were successful in establishing a collaborative and supportive culture. As shown in Fig. 10, mentors observed that the students of 39 teams out 52 teams *evenly* contributed in their respective teams, whilst other teams either were relying on one "leader" to carry out all the tasks (1 team) or had a dominant member who guided the work process and other students contributed unevenly (12 teams). Fig. 11 shows the communication style within the teams in two time-slots (i.e., ideation time and coding time). It shows that the level of the communication within the teams was reasonably high, as over 78.8% of the teams adopted the style where all the students communicated *freely* or *well*,

Table 8
Illustrative quotes for the practices employed by secondary school girls (**N** shows the frequency of each practice that appeared in total in the data).

Theme (practice)	N	Illustrative quotes
Feedback-driven development	**72**	*"We asked [our mentor] to help us figure out what was not working in the code and she really helped us link everything together."* ***SS321*** *"We discussed in a team the possible ways we could overcome the issues so that we all had different ideas. We also talked to our tribe about these issues to get other people's opinions."* ***SS212*** *"The team talked with one another and provided comments on each other's ideas."* ***MO23***
Establishing a collaborative and supportive culture within the team *Highly collaborative team* **(45)** *Group decision making* **(8)** *Voting for hard decisions* **(5)** *Rationalising each decision* **(5)**	**63**	*"With the obstacles [that] we had to overcome along the way, we're a strong team with a great work ethic and I could see this project going beyond a prototype."* ***SS81*** *"The team fixes issues in a collaborative manner whereby each member provides constant input on possible solutions, which is then explored by the entire team."* ***MO8*** *"They thought critically for some justifications for their cost problem and were able to find a way for their justification. Each decision they made was always discussed among each other."* ***MO28***
Simple design, better code *Simplifying the initial design/prototype* **(25)** *Being incremental and iterative* **(11)** *Engaging all team members in coding* **(4)** *Reviewing each other's code* **(3)**	**43**	*"After solving the problems, they sought to further improve the product and patch up any inconsistencies. There was a lot of trial-and-error with the coding. There was evidence of iterative approaches to the problems."* ***MO17*** *"[Our solution was to] work together and checking each other's' code."* ***SS522*** *"They tended to keep it pretty simple and only showed a part of the end idea (app)."* ***MS11***
Predictive thinking	**30**	*"They [team] brainstormed on what they all thought should be included [in the final design] on the whiteboard, so they could visualise and discuss much clearer. Based on this, they were able to draw up what they thought they wanted the final design to look like."* ***MO5*** *"[They] kept brainstorming and spending time developing the ideas and making sure that it was as strong as it could be."* ***MO46*** *"We did not find many challenges as we chose an idea that would work with the micro:bit."* ***SS524***
Leveraging external resources *Utilising online learning resources* **(22)** *Reusing existing codes* **(6)**	**28**	*"I also decided to research some tutorials or general coding techniques to use and asked our mentor for some advice when needed."* ***SS161*** *"[We were] using the website / pre-workshop guidelines to find connections."* ***SS264*** *"[They] looked up online articles and copied code to see if it worked."* ***MO27***
Prioritising quantity over quality	**17**	*"To solve a problem, we talked as a group and tried our best to give as much as ideas and pick the most suitable one for the issue."* ***SS151*** *"We asked questions and also tried different codes over and over again."* ***SS124*** *"[The team was] thinking of possible ways to make it as realistic as possible."* ***MS16***

in which critical evaluations, objections, critiques, and opinions were freely exchanged within the teams. According to Fig. 11, the level of communication increased between the ideation time and the coding time.

We found a number of statements from students and mentors about the effects of group decision-making in addressing the encountered issues. Voting was deemed by a few participants as an effective practice to deal with hard decisions (e.g., *"[The team] narrowed ideas down via voting"* ***MO1***). Some teams tried to rationalise each decision and incorporate all team members' opinions in the decisions made.

Simple design, better code. Throughout the workshop day, students learned that most of the difficulties that they faced with "incorporating ideas into the micro:bit" and "testing and debugging the code" (See Fig. 8) could be addressed by simplifying the initial design and incrementally adding new functionality into the code. Our participants reported that it was originally challenging to incorporate the micro:bit in the original design (prototype); however, after simplifying the design, students were able to use the micro:bit effectively. One student put it as:

> *"We asked ourselves if certain things were necessary for the prototype or if we should just take it to the basics to show our idea and then in further development, we can add more detail."* ***SS91***

Having a simple design and being incremental and iterative enabled students to write code that was more manageable and testable (e.g., *"Instead of trying to replicate the whole app on the micro:bit, [the team] showing just a few functionalities on the micro:bit, [which] made the code a lot more manageable"* ***MO24***). According to **MO17**, this method of problem-solving was employed by her team to track down the root cause of errors. The team was able to resolve the logical errors in their code by breaking the code down into smaller components and testing each of them individually. We found that a few teams could fix errors in the code by engaging all team members in coding and reviewing each other's code. This can be exemplified by the following quote:

> *"We made sure that everyone in our team got to code so that we could each figure out a way to fix."* ***SS473***

Predictive thinking. The challenges faced by students when incorporating their ideas (prototypes) into the micro:bit device taught them that they need to spend more time reflecting on their ideas and predicting the potential outcomes of their ideas before actually implementing the ideas with the micro:bit (Falloon, 2016) (e.g., *"It [our technique] was experimenting with the different blocks and predicting which ones would be applicable to the design that we had envisioned"* ***SS161***). For example, **SS251**, **SS372**, and **SS524** shared that their respective teams did not face any significant challenges during implementation and the reason behind was that they first planned all different ideas concretely and then chose an idea that could be implemented with the micro:bit.

Leveraging external resources. Another notable technique used by students was to browse and leverage external resources when they were unable to solve code-related issues. This practice can also be related to a broader practice in CT called "remixing and reusing" (Brennan and Resnick, 2012). The most commonly used

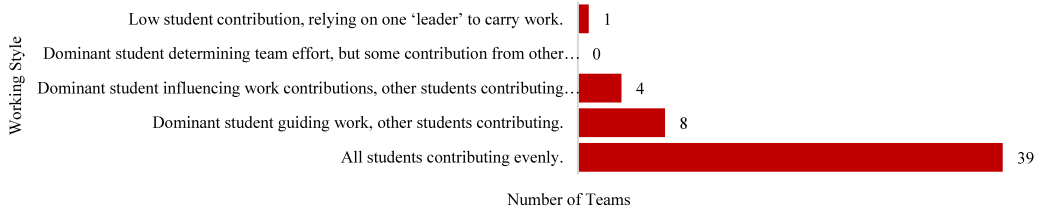

Fig. 10. Teams' working style.

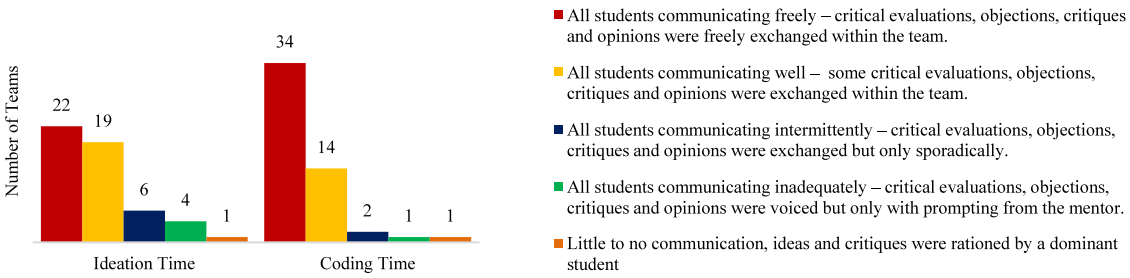

Fig. 11. Teams' communication styles.

resources were micro:bit-related tutorials and websites. For example, **MS20** described that a YouTube tutorial found by her team helped them create an object sensor with the micro:bit. Fewer students leveraged the existing code to add new functionality to their computational solutions. As an example, we have:

> *"Another team member was trying to code by copying the sample code but did not understand how the code works. Once she understood how the code works, she got the code done very quickly."* ***MO3***

Prioritising quantity over quality. We discussed that developing a robust, concrete idea before implementing it with the micro:bit was perceived helpful by some students and mentors to reduce the challenges experienced when transferring ideas to the micro:bit. However, this could lead students to fixate on a few ideas or solutions (Dow et al., 2011, 2010). As a complement practice to "predictive thinking", we observed that students tried to explore the design space of their defined open-ended problems as much as possible and to iteratively diverge and converge the computational ideas and solutions around the problems without deeply thinking about quality, eventually selecting a best-fit one. Indeed, this is often referred by both students and mentors as an effective practice to mainly meet the ideation phase's challenges. One mentor pointed out:

> *"The ideation issues were solved by the other team members and myself encouraging the students to list down problems or solutions regardless of being solvable or easy to implement."* ***MO3***

5 Discussion

CT educational programs are increasingly being designed to increase women's interest, engagement, and participation in computing and technology fields. Challenging stereotypes and structural and cultural factors that negatively shape women's enthusiasm and abilities around computing is an important step in dealing with the representation issues in STEM (Lang et al., 2020; Kong et al., 2018). With this in mind, we designed and conducted a girls-only CT program (i.e., the OzGirlsCT program) with one of the aims being to explore the effect of the OzGirlsCT program on the secondary school girls' CT practices when they do collaborative programming with the micro:bit. The feedback received from the participating girls shows that the OzGirlsCT program was an inspiring, engaging, and impactful education program. For example, one student said, "*I was surprised by how much I positively benefited from this experience and have learned a lot more than I anticipated. I feel inspired*". Another student pointed out, "*The program has really kick-started my interest in design tech. I have always found it fascinating, but now I know coding and technology can really be a possible career in my future*". The literature shows that girls-only programs can be beneficial to girls — increasing their performance and boosting their self-efficacy and confidence when compared to co-educational programs (Ashcraft et al., 2012; Barker and Aspray, 2006). Focusing on girls-only computing programs can lead to improved diversity outcomes in the workforce, make labour markets more competitive, and ensure girls have access to the opportunities available in STEM and Entrepreneurship (Barker and Aspray, 2006; Miliszewska and Sztendur, 2010; Teague, 2002). In the following sections, we first draw some important implications of the findings of this study and the design of the OzGirlsCT program in the context of the literature on women in computing for research and practice. Then, we discuss the limitations of our study.

5.1 Implications

Our study confirmed that debugging is a complex and difficult CT practice. We quantitatively explored the difficulty level of a set of CT practices by gathering the perceptions of 193 secondary school girls and 31 mentors. Our data shows that code debugging measured with the statements "*identifying errors in code*" and "*finding a solution to fix the identified errors in code*" was the most difficult CT practice for the secondary school girls to apply (Finding 2). The following reasons can be attributed to why debugging was the most difficult CT practice for the secondary school girls:

(1) Learning and applying debugging need an adequate amount of time. Many researchers have investigated the role of gender in developing and acquiring CT skills, and the findings in this regard are mixed. Several studies (e.g., Durak and Saritepeci, 2018; Werner et al., 2012) found no significant relationship between CT skills (including debugging) and gender. Regardless of gender and age, it appears that debugging is a complicated CT practice for learners (e.g., Dorn et al., 2018; Aivaloglou and Hermans, 2019), needing a systematic approach and a substantial amount

Finding 8. *We discovered six main practices and techniques that novice girl programmers found to be the most effective practices to deal with the challenges in collaborative ideation and coding. Whilst, we confirm that some of these practices (i.e., "predictive thinking" and "leveraging external resources") are already revealed/discussed by the literature (Zhang and Nouri, 2019) as a computational thinking practice, four of these practices including "feedback-driven development", "establishing a collaborative and supportive culture within the team", "simple design, better code", and "prioritising quantity over quality" have not been frequently and rigorously examined in the computational thinking literature.*

of time to understand the process and develop the necessary skills (Zhang and Nouri, 2019; Wohl et al., 2015; Falloon, 2016). Even for experienced software practitioners, debugging is a complex and time-consuming task (Zeller, 2009; Perscheid et al., 2017). This is supported by the outcome of RQ2 (Finding 4), which has statistically found that, among all CT practices, having prior knowledge and experience in coding can significantly reduce the difficulty level of debugging. Similarly, Wilcox and Lionelle (2018) found that girls gained more benefits than boys from prior coding/computing experience in the introductory computer science course. While the literature findings are mixed about the difficulty level of CT practices for boys and girls, there is stronger support for the significant amount of training time needed to adequately develop and apply some CT skills (Atmatzidou and Demetriadis, 2016; Bers et al., 2014).

(2) Education programs should be supportive of debugging. The organisation and activities of the OzGirlsCT program as an education program may have led to the girls experienced more difficulty in debugging. First, as we followed the problem-based learning approach in the OzGirlsCT program, we did not provide the girls with formal debugging training (See Section 3.1.2). Further to this, the OzGirlsCT program was a one-day program and covered several topics and activities (e.g., brainstorming problems, presenting business pitches). Hence, the girls had limited time to work on their prototypes, identify possible errors in their prototypes (**CTP8**), and find a solution to fix the identified errors (**CTP9**). Finally, as the girls had the freedom to choose and work on their projects, many initially chose projects which were complex. Hence, the complexity of the projects may also have increased the difficulty level of debugging for the girls. Given the nature of the OzGirlsCT program and the minimal level of the students' coding experience, the satisfactory development of a complex CT practice like debugging was going to be a challenge.

(3) Programming learning tools need to provide active help during the debugging process. If we accept that debugging is a complex CT practice, programming learning tools and environments are expected to provide scaffolding and mechanisms to guide learners through this complexity (Webb and Rosson, 2013; Denner et al., 2012). However, it seems that most environments (including the micro:bit device which our program used) do not provide guidance, visual clues, and features by which (young) learners can understand the nature of an error, the reason behind it, and the path to resolution (Jordan-Douglass et al., 2018). As described in Section 4.4, the girls' challenges when working on their prototypes can also be attributed to the micro:bit limitations. The girls frequently indicated that the micro:bit device did not have some of the capabilities and features to support them in incorporating, implementing, and debugging their ideas. In Aivaloglou and Hermans (2019), a survey of 98 teachers who taught programming in code clubs with different languages, such as Scratch, Python, micro:bit, Java, found that debugging was the most commonly reported learning barrier for boys and girls. The study by Wohl et al. (2015) observed that girls and boys could better understand the concept of debugging with unplugged and Cubelets sessions than when they used Scratch (a similar learning environment to the micro:bit device). It is because the physicality of unplugged and Cubelets sessions helped students understand why an error happened, while the errors produced in Scratch only showed that the program did not work.

All this indicates that debugging is an especially difficult CT practice and may require specialised and customised educational materials and tools for girls (Perscheid et al., 2017; Grigoreanu et al., 2006). We propose that learning tools and environments and education programs need to empower girls in facing debugging challenges and guide them through the debugging process. According to Zeller (2009), the debugging process consists of seven steps: (1) "track the problem"; (2) "reproduce the problem"; (3) "automate and simplify the test case"; (5) "find possible infection origins"; (6) "focus on the most likely origins"; (6) "isolate the infection chain"; and (7) "correct the defect". In this study, we only focused on two steps of debugging (i.e., locating and correcting faults). Hence, there is a need for a deep investigation to understand how girls perceive each step of the debugging process (Grigoreanu et al., 2006) and how learning tools and environments can support girls through each step. We assert that educational researchers and practitioners should focus on designing CT learning tools and programs that are an ongoing element of the curriculum, much like mathematics, and are introduced early in the education system, as our findings show that experience helps girls deal with complex CT practices such as debugging.

Our study highlighted the ease with which secondary school girls applied the collaborative aspects of CT. Collaboration is a key competence in the 21st century (Voogt and Roblin, 2012). We measured quantitatively the secondary school girls' attitude toward collaborative problem-solving with three statements: "*giving feedback to teammates and making suggestions to improve idea/code*", "*working collaboratively with team members*", and "*reaching a consensus in group decisions*". We have learned that the collaborative practices of CT were the easiest practices for the secondary school girls to apply (Finding 3). Studies show that gender may influence the attitude toward and the outcome of collaborative problem-solving (OECD, 2017; Apesteguia et al., 2012). Ardito et al. (2020) found that sixth-grade girls concentrate more on group dynamics and metacognitive process when developing and applying CT skills, while boys are more engaged with the operational aspects of building and coding. They observed that girls in sixth grade scored highest in "teamwork/leadership/effective communication" (analogue to collaboration). Our qualitative analysis also confirms Finding 3 and the observation of Ardito et al. (2020). We found having a collaborative and supportive team and seeking feedback from peers and mentors were deemed by the secondary school girls as effective practices in addressing some technical challenges. These two practices also positively impacted other practices employed by the girls in dealing with technical challenges (See Fig. 8 and Fig. 9). Jun and colleagues in Jun et al. (2017) did not take into account gender in CT skills development. They found that among four steps (i.e., design, personalisation, collaboration, and reflection) of design-based learning (DBL), reflection was the most difficult step for elementary school students, followed by collaboration. It was also revealed that in the traditional, direct method, collaboration was the most difficult step. Interestingly, Jun et al. (2017) revealed that collaboration was the most popular step among elementary school students in DBL methodology.

Our findings are aligned with the studies (e.g., Ashcraft et al., 2012; Liebenberg et al., 2012; Buffum et al., 2016; McDowell et al., 2002) showing the benefits of incorporating collaboration in CT learning environments, which can help girls deal with complex CT practices and advanced programming tasks. McDowell et al. (2002) showed that collaboration could increase girls' persistence in the debugging process. In another study, Buffum et al. (2016) and Liebenberg et al. (2012) indicated that collaborative learning helped increase girls' enjoyment in CT. The study of Buffum et al. also showed that when girls were paired with a more experienced boy, they were able to solve challenging programming problems as much as the boys. Given the positive influences of collaboration on girls in developing and applying CT skills and our findings regarding the secondary school girls' attitudes toward collaborative problem-solving, we assert that future CT and computer science learning programs for girls should emphasise and encourage collaboration in the program design.

The friendship level of the girls in the teams may have also contributed to the results obtained for the collaborative practices of CT. As discussed in Section 3.1.1.1, while the friendship level of the team members in the teams varied widely, some teams had members who were close friends. Hence, it was not difficult for such teams to work collaboratively on their computational solutions and reach a common goal under pressure conditions. We suggest that the results of the collaborative practices of CT may have been different if the students did not know each other. This necessitates a replication study to investigate how girls who do not know each other perceive the collaborative practices of CT. Finally, further research should focus on possible differences in girl-only, boy-only, and mixed-gender teams in collaborative ideation and programming and in perceiving the difficulty level of the collaborative practices of CT.

Our study found that focusing on collaborative, creative problem-solving for real-world problems was a strong motivator for our student teams. Many introductory CT programs focus on programming for pre-defined problems or canned exercises, which may prevent young girls from understanding how computing and technology can help address realistic problems in society (Lang et al., 2020). We designed our program with the view of providing opportunities that allowed secondary school girls to collaboratively identify and work on complex problems that are important for them and their society (i.e., real word problems). Working collaboratively on computational projects in personal and social contexts is considered a powerful approach to engage underrepresented populations (e.g., young girls) in computing fields (Brady et al., 2017; Pinkard et al., 2017; Tissenbaum et al., 2019; Kafai and Burke, 2013). In the context of CT, this trend is sometimes called "computational action" (Tissenbaum et al., 2019). Given this opportunity in our program, the secondary school girls developed creative computational solutions to a wide range of personally and socially relevant, realistic problems such as "*how to improve the safety of cyclists*", "*how to reduce food wastage*", "*how to help beginners grow sustainable produce in their own home or garden*", and "*how to help pet owners to take care of pets while they are away*". This process included creative thinking, risk-taking, collaborative problem-solving, and hands-on experiences. The teams collaboratively developed their computational ideas and collected requirements to formulate, implement, and evaluate ideas with the micro:bit. This type of activity positively influences girls' interests in pursuing computing (Ashcraft et al., 2012).

Observations of our program indicate that exposing the secondary school girls to complex and realistic problems initially puts them in a relatively difficult situation, and appropriate, relevant instruction is required. Their level of coding experience, together with the limitations of the micro:bit device, made it difficult for the girls to implement and debug their sometimes complex ideas with the micro:bit. That is why we found many references in our qualitative data relating to the difficulty of incorporating their idea into the micro:bit device (Finding 6). In Section 4.4, we argued that this challenge mainly stems from the difficulty in applying three CT practices: decomposition ("breaking down or narrowing an idea"), abstraction ("leaving out irrelevant/unimportant stuff from an idea"), and algorithm ("creating a series of ordered steps to implement an idea"), in which the difficulty level of abstraction was the second highest amongst our participants (Finding 2). Similarly, other studies (Webb and Rosson, 2013; Hoover et al., 2016) observed that middle school girls faced barriers in understanding and applying abstraction. Interestingly, the difficulty of incorporating the team's idea into the micro:bit device gradually taught our participating girls that they need to simplify their design, leave out unrelated materials in their initial ideas, and only prototype their solutions with the micro:bit. We referred to this practice in Section 4.5 as "simple design, better code", which was deemed an effective practice to address the challenges related to "incorporating ideas into the micro:bit", "code debugging", and "code complexity" faced by the secondary school girls (See Fig. 8). We suggest while developing and working on computational ideas for real-world problems is an excellent motivator and should be considered a strength of CT programs, care should be taken to ensure that young girls understand that they need to 'walk before they can run'. They need to prototype their computational ideas before attempting full implementation, as facing complex programming challenges (e.g., debugging a complex computational solution) early on in the process can be demotivating, and significantly hinder progress, stymieing positive thoughts about courses and careers in computing.

5.2 Limitations

Our findings in this study could be influenced by our sampling method. This study focused only on girls from 44 secondary schools in the state of Victoria in Australia. Further, most of our participating students were self-reported high-performers with good academic records. Almost half of them also had at least one month of coding experience before the OzGirlsCT program (i.e., See Fig. 5). While an important body of literature (e.g., Aivaloglou and Hermans, 2019; Atmatzidou and Demetriadis, 2016; Durak and Saritepeci, 2018; Ardito et al., 2020) shows significant similarities between boys and girls in terms of developing and applying CT practices in different programming languages, our findings are exclusive to secondary school girls with good academic records and prior coding experience. Hence, our study cannot claim that the findings and recommendations be easily transferred or applicable to primary school girls, a broader class of secondary school girls, or students of any gender. Recruiting the participants likely had self-selection bias (Lavrakas, 2008), as the girls and mentors who chose to participate were more likely interested in the study topic. Our findings may be different if this study is replicated in different contexts (e.g., replicating the study with the secondary school girls who do not know each other or are forced to participate in the study regardless of interest). Despite these limitations, we strongly believe that the design of our study and the OzGirlsCT program is not restricted to a particular context, which will facilitate the replication and validation of this study and the OzGirlsCT program in different contexts such as male students or students in different countries.

Another possible limitation emerges from the structure and formulation of the questions used in the surveys and observation study (Litwin, 1995; Sigelaman, 1981). We formulated the questions used in this study based on a review of formal and grey

literature and our prior experience in software engineering research. Furthermore, the questions were fine-tuned and validated through several internal discussions among authors and other colleagues. While we are confident that our questions covered important CT practices that can be introduced and evaluated through programming in K-12 settings, we confirm that some CT practices such as modularity and creativity have not been covered.

The analysis and classification of the qualitative data (i.e., the answers to the open-ended questions) were mainly performed by the first author. Whilst Tómasdóttir et al. (2017) argue that this type of analysis process can help increase consistency in the findings, this can be another source of threat to the findings of this study. Our approach to moderate this threat was discussing and cross-validating the categories and their respective quotes with other researchers involved in this study, particularly with the second author. We also minimised the subjective judgement in the findings by reporting those findings that have appeared in more than one data source and reported by multiple participants. Despite these efforts, we cannot claim that we could completely exclude the possibility of interpretation and classification errors.

In this study, the mentors played two roles: facilitator and observer. One potential threat here is that the mentors may have become biased by taking on the role as a member of their teams and becoming involved in the problem-solving process of their teams. Our strategy to deal with this threat was to organise a training workshop for the mentors to clearly define their roles and responsibilities during the OzGirlsCT workshops (See Section 3.1.1.2). Furthermore, during the OzGirlsCT workshops, we occasionally reminded the mentors that they should not be directly involved in the problem-solving process of their teams. Despite these mitigating strategies, it is possible that being an observer may have interfered with their mentoring activities and contributed to the team's frustration, as described in Section 4.4, as they were not as readily available to help.

Social desirability bias (Furnham, 1986) (i.e., a participant tends to provide socially acceptable answers) may have influenced the students' and mentors' responses. The potential social desirability bias in the students' responses was mitigated to some extent by asking the mentors the same questions as those of the students. To further reduce this bias, we assured the participants that their personal information and answers would not be identified in any potential reports in the future.

6 Summary

This study empirically explored how secondary school girls perceive Computational Thinking (CT) practices by conducting a mixed-methods approach consisting of two surveys with secondary school girls (with 193 valid responses each), a survey with 31 valid responses collected from 31 mentors, and 52 mentor observation reports. Our goal was to understand how secondary school girls perceive the difficulty level of 12 CT practices when developing and implementing computational solutions to socially relevant problems with the micro:bit device in a collaborative setting. We were also interested in understanding the challenges they faced in programming with the micro:bit device in this setting and the best practices they developed and applied to address these challenges.

(1) The quantitative analysis shows that "*identifying errors in code*" (28% of the girls and 42% of the mentors rated it *difficult* or *very difficult*) and "*finding a solution to fix the identified errors in code*" (29% of the girls and 32% of the mentors rated it *difficult* or *very difficult*) are the most difficult CT practices to apply. In contrast, the collaborative practices of CT represented by "*reaching a consensus in group decisions*", "*working collaboratively with team members*", and "*giving feedback to teammates and making suggestions to improve idea/code*" are the easiest practices to apply, as in each case, at least 67% of the participants believe that they are (very) easy practices to apply.

(2) The challenges that the girls faced when developing and implementing their computational solutions with the micro:bit device are not *only* technical (i.e., challenges related to "incorporating idea into the micro:bit", "code debugging", and "code complexity"), but also are rooted in "the micro:bit limitations", "personality traits", and "coding experience".

(3) Whilst having prior experience in coding to some extent influences the difficulty level of CT practices perceived by the girls, its most significant effect is on "debugging".

(4) The main practices employed by the girls to overcome the challenges faced are "feedback-driven development", "establishing a collaborative and supportive culture within the team", "simple design, better code", "predictive thinking", "prioritising quantity over quality", and "leveraging external resources".

CRediT authorship contribution statement

Mojtaba Shahin: Conceptualisation, Methodology, Investigation, Formal analysis, Validation, Writing – original draft, Writing – review & editing. **Christabel Gonsalvez:** Conceptualisation, Methodology, Investigation, Validation, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Jon Whittle:** Conceptualisation, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Chunyang Chen:** Conceptualisation, Methodology, Writing – review & editing, Supervision. **Li Li:** Conceptualisation, Methodology, Writing – review & editing, Supervision. **Xin Xia:** Conceptualisation, Methodology, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is funded by the Australian Government, Department of Industry, Innovation and Science, Grant No. WISE64905. The authors would like to thank all participants in this study.

References

Aho, A.V., 2012. Computation and computational thinking. Comput. J. 55 (7), 832–835. http://dx.doi.org/10.1093/comjnl/bxs074.

Aivaloglou, E., Hermans, F., 2019. How is programming taught in code clubs? Exploring the experiences and gender perceptions of code club teachers. In: Proceedings of the 19th Koli Calling International Conference on Computing Education Research. pp. 1–10.

Alvarado, C., Umbelino, G., Minnes, M., 2018. The persistent effect of pre-college computing experience on college CS course grades. In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education. pp. 876–881.

Angeli, C., Valanides, N., 2020. Developing young children's computational thinking with educational robotics: An interaction effect between gender and scaffolding strategy. Comput. Hum. Behav. 105, 105954.

Angevine, C., Cator, K., Roschelle, J., Thomas, S.A., Waite, C., Weisgrau, J., 2017. Computational thinking for a computational world.

Apesteguia, J., Azmat, G., Iriberri, N., 2012. The impact of gender composition on team performance and decision making: Evidence from the field. Manage. Sci. 58 (1), 78–93.

Ardito, G., Czerkawski, B., Scollins, L., 2020. Learning computational thinking together: Effects of gender differences in collaborative middle school robotics program. TechTrends 1–15.

Ashcraft, C., Eger, E., Friend, M., 2012. Girls in IT: The Facts. National Centre for Women & Information Technology (NCWIT), Boulder, CO, Retrieved July 2013, ed.

Astrachan, O., Barnes, T., Garcia, D.D., Paul, J., Simon, B., Snyder, L., 2011. CS principles: piloting a new course at national scale. In: Presented at the 42nd ACM Technical Symposium on Computer Science Education.

Atmatzidou, S., Demetriadis, S., 2016. Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. Robot. Auton. Syst. 75, 661–670. http://dx.doi.org/10.1016/j.robot.2015.10.008.

Ball, T., et al., 2016. Microsoft touch develop and the BBC micro: bit. In: Presented at the IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C).

Barker, L.J., Aspray, W., 2006. The State of Research on Girls and IT. na.

Barr, V., Stephenson, C., 2011. Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? Inroads 2 (1), 48–54.

Beller, M., 2018. Toward an empirical theory of feedback-driven development. In: Presented at the 40th International Conference on Software Engineering: Companion Proceeedings. Gothenburg, Sweden.

Bereiter, C., Scardamalia, M., 1989. Intentional learning as a goal of instruction. In: Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser. pp. 361–392.

Berl, M., Lee, V.R., 2011. Collaborative strategic board games as a site for distributed computational thinking. Int. J. Game-Based Learn. (IJGBL) 1 (2), 65–81.

Bers, M.U., Flannery, L., Kazakoff, E.R., Sullivan, A., 2014. Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. Comput. Educ. 72, 145–157.

Bocconi, S., et al., 2016. Developing Computational Thinking in Compulsory Education. European Commission, JRC Science for Policy Report.

Boddington, M., Barakat, S., 2018. Exploring alternative gendered social structures within entrepreneurship education: notes from a women's-only enterprise program in the United Kingdom. In: Women Entrepreneurs and the Myth of 'Underperformance'. Edward Elgar Publishing.

Botella, C., Rueda, S., López-Iñesta, E., Marzal, P., 2019. Gender diversity in STEM disciplines: A multiple factor problem. Entropy 21 (1), 30.

Brackmann, C.P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., Barone, D., 2017. Development of computational thinking skills through unplugged activities in primary school. In: Presented at the 12th Workshop on Primary and Secondary Computing Education. Nijmegen, Netherlands.

Brady, C., Orton, K., Weintrop, D., Anton, G., Rodriguez, S., Wilensky, U., 2017. All roads lead to computing: Making, participatory simulations, and social computing as pathways to computer science. IEEE Trans. Educ. 60 (1), 59–66. http://dx.doi.org/10.1109/TE.2016.2622680.

Brennan, K., Resnick, M., 2012. New frameworks for studying and assessing the development of computational thinking. In: Presented at the 2012 Annual Meeting of the American Educational Research Association. Vancouver, Canada.

Buffum, P.S., Frankosky, M., Boyer, K.E., Wiebe, E.N., Mott, B.W., Lester, J.C., 2016. Collaboration and gender equity in game-based learning for middle school computer science. Comput. Sci. Eng. 18 (2), 18–28.

Cohen, B.H., 2008. Explaining Psychological Statistics. John Wiley & Sons.

Cuny, J., Snyder, L., Wing, J.M., 2010. Demystifying computational thinking for non-computer scientists- Unpublished manuscript in progress. [Online]. Available: http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

Debnath, M., Pandey, M., Chaplot, N., Gottimukkula, M.R., Tiwari, P.K., Gupta, S.N., 2012. Role of soft skills in engineering education: students' perceptions and feedback. In: Nair, C.S., Patil, A., Mertova, P. (Eds.), Enhancing Learning and Teaching Through Student Feedback in Engineering. Chandos Publishing, pp. 61–82.

Denner, J., Werner, L., Campe, S., Ortiz, E., 2014. Pair programming: Under what conditions is it advantageous for middle school students? J. Res. Technol. Educ. 46 (3), 277–296. http://dx.doi.org/10.1080/15391523.2014.888272.

Denner, J., Werner, L., Ortiz, E., 2012. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? Comput. Educ. 58 (1), 240–249.

Devine, J., Finney, J., de Halleux, P., Moskal, M., Ball, T., Hodges, S., 2019. MakeCode and CODAL: intuitive and efficient embedded systems programming for education. J. Syst. Archit. 98, 468–483.

Doleck, T., Bazelais, P., Lemay, D.J., Saxena, A., Basnet, R.B., 2017. Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: exploring the relationship between computational thinking skills and academic performance. J. Comput. Educ. 4 (4), 355–369. http://dx.doi.org/10.1007/s40692-017-0090-9.

Dorn, N., Berges, M., Capovilla, D., Hubwieser, P., 2018. Talking at cross purposes: perceived learning barriers by students and teachers in programming education. In: Proceedings of the 13th Workshop in Primary and Secondary Computing Education. pp. 1–4.

Dow, S., Fortuna, J., Schwartz, D., Altringer, B., Schwartz, D., Klemmer, S., 2011. Prototyping dynamics: sharing multiple designs improves exploration, group rapport, and results. In: Presented at the ACM Conference on Human Factors in Computing Systems. Vancouver, BC, Canada.

Dow, S.P., Glassco, A., Kass, J., Schwarz, M., Schwartz, D.L., Klemmer, S.R., 2010. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. ACM Trans. Comput.-Hum. Interact. 17 (4), 1–24. http://dx.doi.org/10.1145/1879831.1879836.

Durak, H.Y., Saritepeci, M., 2018. Analysis of the relation between computational thinking skills and various variables with the structural equation model. Comput. Educ. 116, 191–202. http://dx.doi.org/10.1016/j.compedu.2017.09.004.

Easterbrook, S., Singer, J., Storey, M.-A., Damian, D., 2008. Selecting empirical methods for software engineering research. In: Shull, F., Singer, J., Sjøberg, D.I.K. (Eds.), Guide to Advanced Empirical Software Engineering. Springer London, London, pp. 285–311.

Falloon, G., 2016. An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad. J. Comput. Assist. Learn. 32 (6), 576–593. http://dx.doi.org/10.1111/jcal.12155.

Fawcett, L.M., Garton, A.F., 2005. The effect of peer collaboration on children's problem-solving ability. Br. J. Educ. Psychol. 75 (2), 157–169.

Feldhausen, R., Weese, J.L., Bean, N.H., 2018. Increasing student self-efficacy in computational thinking via STEM outreach programs. In: Presented at the 49th ACM Technical Symposium on Computer Science Education. Baltimore, Maryland, USA.

Fessard, G., Wang, P., Renna, I., 2019. Are there differences in learning gains when programming a tangible object or a simulation? In: ACM Conference on Innovation and Technology in Computer Science Education. ACM, Aberdeen, Scotland UK, pp. 78–84.

Field, A., 2013. Discovering Statistics using IBM SPSS Statistics. SAGE Publications Inc..

Furnham, A., 1986. Response bias, social desirability and dissimulation. Personal. Individ. Differ. 7 (3), 385–400. http://dx.doi.org/10.1016/0191-8869(86)90014-0.

Gabay-Egozi, L., Shavit, Y., Yaish, M., 2014. Gender differences in fields of study: The role of significant others and rational choice motivations. Eur. Sociol. Rev. 31 (3), 284–297.

Glaser, B.G., Strauss, A.L., 2017. Discovery of Grounded Theory: Strategies for Qualitative Research. Routledge.

Grigoreanu, V., et al., 2006. Gender differences in end-user debugging, revisited: What the miners found. In: Visual Languages and Human-Centric Computing (VL/HCC'06). IEEE, pp. 19–26.

Grover, S., Pea, R., 2013. Computational thinking in K–12: A review of the state of the field. Educ. Res. 42 (1), 38–43. http://dx.doi.org/10.3102/0013189X12463051.

Grover, S., Pea, R., Cooper, S., 2015. Designing for deeper learning in a blended computer science course for middle school students. Comput. Sci. Educ. 25 (2), 199–237. http://dx.doi.org/10.1080/08993408.2015.1033142.

Günbatar, M.S., 2019. Computational thinking within the context of professional life: Change in CT skill from the viewpoint of teachers. Educ. Inf. Technol. 24 (5), 2629–2652. http://dx.doi.org/10.1007/s10639-019-09919-x.

Hmelo-Silver, C.E., 2004. Problem-based learning: What and how do students learn? Educ. Psychol. Rev. 16 (3), 235–266.

Hoda, R., 2011. Self-Organizing Agile Teams: A Grounded Theory (Ph.D. thesis). Victoria University of Wellington.

Hoda, R., Noble, J., 2017. Becoming agile: A grounded theory of agile transitions in practice. In: IEEE/ACM 39th International Conference on Software Engineering (ICSE). Buenos Aires, Argentina 20-28 May 2017. pp. 141–151. http://dx.doi.org/10.1109/ICSE.2017.21.

Hoover, A.K., et al., 2016. Assessing computational thinking in students' game designs. In: Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts. ACM, Austin, Texas, USA, pp. 173–179. http://dx.doi.org/10.1145/2968120.2987750, 2987750.

Howell, D.C., 2012. Statistical Methods for Psychology. Cengage Learning.

Hsu, T.-C., Chang, S.-C., Hung, Y.-T., 2018. How to learn and how to teach computational thinking: Suggestions based on a review of the literature. Comput. Educ. 126, 296–310. http://dx.doi.org/10.1016/j.compedu.2018.07.004.

Hung, W., 2006. The 3C3R model: A conceptual framework for designing problems in PBL. Interdiscip. J. Probl.-Based Learn. 1 (1), 6.

Jordan-Douglass, A., Kumar, V., Woods, P.J., 2018. Exploring computational thinking through collaborative problem solving and audio puzzles. In: 17th ACM Conference on Interaction Design and Children. ACM, Trondheim, Norway, pp. 513–516. http://dx.doi.org/10.1145/3202185.3210766, 3210766.

Jun, S., Han, S., Kim, S., 2017. Effect of design-based learning on improving computational thinking. Behav. Inf. Technol. 36 (1), 43–53.

K-12 Computer Science Framework Steering Committee, 2016. K-12 Computer Science Framework. ACM, ISBN: 978-1-4503-5278-9, [Online]. Available: https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf.

Kafai, Y., Burke, Q., 2013. The social turn in K-12 programming: moving from computational thinking to computational participation. In: 44th ACM Technical Symposium on Computer Science Education. ACM, Denver, Colorado, USA, pp. 603–608. http://dx.doi.org/10.1145/2445196.2445373, 2445373.

Kafai, Y.B., Resnick, M., 2012. Constructionism in Practice: Designing, Thinking, and Learning in a Digital World. Routledge.

Kazimoglu, C., Kiernan, M., Bacon, L., MacKinnon, L., 2012. Learning programming at the computational thinking level via digital game-play. Procedia Comput. Sci. 9, 522–531.

Kelion, L., 2021. Seven outstanding Micro Bit projects. https://www.bbc.com/news/technology-35824446 (accessed 10 June, 2021).

Kesar, S., 2017. Closing the STEM Gap: Why STEM Classes and Career Still Lack Girls and What We Can Do About It. Microsoft, [Online]. Available: https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE1UMWz.

Ketenci, T., Calandra, B., Margulieux, L., Cohen, J., 2019. The relationship between learner characteristics and student outcomes in a middle school computing course: An exploratory analysis using structural equation modeling. J. Res. Technol. Educ. 51 (1), 63–76.

Kirkpatrick, M.S., Mayfield, C., 2017. Evaluating an alternative CS1 for students with prior programming experience. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. pp. 333–338.

Kong, S.-C., 2019. Components and methods of evaluating computational thinking for fostering creative problem-solvers in senior primary school education. In: Kong, S.-C., Abelson, H. (Eds.), Computational Thinking Education. Springer Singapore, Singapore, pp. 119–141.

Kong, S.-C., Abelson, H., Lai, M., 2019. Introduction to computational thinking education. In: Kong, S.-C., Abelson, H. (Eds.), Computational Thinking Education. Springer Singapore, Singapore, pp. 1–10.

Kong, S.-C., Chiu, M.M., Lai, M., 2018. A study of primary school students' interest, collaboration attitude, and programming empowerment in computational thinking education. Comput. Educ. 127, 178–189.

Korkmaz, Ö., Çakir, R., Özden, M.Y., 2017. A validity and reliability study of the computational thinking scales (CTS). Comput. Hum. Behav. 72, 558–569. http://dx.doi.org/10.1016/j.chb.2017.01.005.

Lang, C., Fisher, J., Craig, A., Forgasz, H., 2020. Computing, girls and education: What we need to know to change how girls think about information technology. Australas. J. Inf. Syst. 24.

Lavrakas, P.J., 2008. Encyclopedia of Survey Research Methods. Sage Publications.

Lewis, C.M., 2012. The importance of students' attention to program state: a case study of debugging behavior. In: Proceedings of the Ninth Annual International Conference on International Computing Education Research. pp. 127–134.

Liebenberg, J., Mentz, E., Breed, B., 2012. Pair programming and secondary school girls' enjoyment of programming and the subject Information Technology (IT). Comput. Sci. Educ. 22 (3), 219–236.

Litwin, M.S., 1995. How to Measure Survey Reliability and Validity. Sage Publications.

Lye, S.Y., Koh, J.H.L., 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? Comput. Hum. Behav. 41, 51–61. http://dx.doi.org/10.1016/j.chb.2014.09.012.

Malhotra, N., Birks, D., 2007. Marketing Research: An Applied Approach: 3rd European Edition. Pearson education.

McCrum-Gardner, E., 2008. Which is the correct statistical test to use? Br. J. Oral Maxillofac. Surg. 46 (1), 38–41. http://dx.doi.org/10.1016/j.bjoms.2007.09.002.

McDowell, C., Werner, L., Bullock, H., Fernald, J., 2002. The effects of pair-programming on performance in an introductory programming course. In: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education. pp. 38–42.

Meade, A.W., Craig, S.B., 2012. Identifying careless responses in survey data. Psychol. Methods 17 (3), 437.

2021. Micro:bit educational foundation. https://microbit.org/ (accessed 10 June, 2021).

Miliszewska, I., Sztendur, E.M., 2010. Interest in ICT studies and careers: Perspectives of secondary school female students from low socioeconomic backgrounds. Interdiscip. J. Inf. Knowl. Manage. 5, 237–260.

National Research Council, 2010. Report of a Workshop on the Scope and Nature of Computational Thinking. National Academies Press.

Nordstokke, D.W., Zumbo, B.D., 2010. A new nonparametric Levene test for equal variances. Psicológica 31 (2), 401–430.

Nordstokke, D.W., Zumbo, B.D., Cairns, S.L., Saklofske, D.H., 2011. The operating characteristics of the nonparametric levene test for equal variances with assessment and evaluation data. Pract. Assess. Res. Eval. 16.

OECD, 2017. PISA 2015 Results (Volume V). PISA, OECD Publishing.

Papert, S., 1980. Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, Inc..

Perscheid, M., Siegmund, B., Taeumel, M., Hirschfeld, R., 2017. Studying the advancement in debugging practice of professional software developers. Softw. Qual. J. 25 (1), 83–110.

Pinkard, N., Erete, S., Martin, C.K., McKinney de Royston, M., 2017. Digital youth divas: Exploring narrative-driven curriculum to spark middle school girls' interest in computational activities. J. Learn. Sci. 26 (3), 477–516. http://dx.doi.org/10.1080/10508406.2017.1307199.

Resnick, M., et al., 2009. Scratch: Programming for all. Commun. ACM 52 (11), 60–67.

Rodriguez, B., Kennicutt, S., Rader, C., Camp, T., 2017. Assessing computational thinking in CS unplugged activities. In: ACM SIGCSE Technical Symposium on Computer Science Education. ACM, Seattle, Washington, USA, pp. 501–506. http://dx.doi.org/10.1145/3017680.3017779, 3017779.

Román-González, M., Pérez-González, J.-C., Moreno-León, J., Robles, G., 2018. Extending the nomological network of computational thinking with non-cognitive factors. Comput. Hum. Behav. 80, 441–459. http://dx.doi.org/10.1016/j.chb.2017.09.030.

Sadler, P.M., Sonnert, G., Hazari, Z., Tai, R., 2012. Stability and volatility of STEM career interest in high school: A gender study. Sci. Educ. 96 (3), 411–427. http://dx.doi.org/10.1002/sce.21007.

Savery, J.R., 2015. Overview of problem-based learning: Definitions and distinctions. In: Essential Readings in Problem-Based Learning. Purdue University Press, pp. 5–15.

Schöpfel, J., Farace, D.J., 2009. Grey literature. In: Encyclopedia of Library and Information Sciences. CRC Press, pp. 2029–2039.

Schwarz, N., Oyserman, D., 2001. Asking questions about behavior: Cognition, communication, and questionnaire construction. Am. J. Eval. 22 (2), 127–160. http://dx.doi.org/10.1177/109821400102200202.

Scott, A.J., Knott, M., 1974. A cluster analysis method for grouping means in the analysis of variance. Biometrics 507–512.

Seneviratne, O., 2017. Making computer science attractive to high school girls with computational thinking approaches: A case study. In: Emerging Research, Practice, and Policy on Computational Thinking. Springer, pp. 21–32.

Sengupta, P., Kinnebrew, J.S., Basu, S., Biswas, G., Clark, D., 2013. Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. Educ. Inf. Technol. 18 (2), 351–380. http://dx.doi.org/10.1007/s10639-012-9240-x.

Shahin, M., Ilic, O., Gonsalvez, C., Whittle, J., 2021. The impact of a STEM-based entrepreneurship program on the entrepreneurial intention of secondary school female students. Int. Entrep. Manage. J. 1–32.

Sherman, M., Martin, F., 2015. The assessment of mobile computational thinking. J. Comput. Sci. Coll. 30 (6), 53–59.

Shull, F., Singer, J., Sjøberg, D.I.K., 2007. Guide to Advanced Empirical Software Engineering. Springer-Verlag.

Shute, V.J., Sun, C., Asbell-Clarke, J., 2017. Demystifying computational thinking. Educ. Res. Rev. 22, 142–158. http://dx.doi.org/10.1016/j.edurev.2017.09.003.

Sigelaman, L., 1981. Question-order effects on presidential popularity. Public Opin. Q. 45 (2), 199–207.

Snalune, P., 2015. The benefits of computational thinking. ITNOW 57 (4), 58–59. http://dx.doi.org/10.1093/itnow/bwv111.

Soppe, M., Schmidt, H.G., Bruysten, R.J., 2005. Influence of problem familiarity on learning in a problem-based course. Instr. Sci. 33 (3), 271–281.

Spieler, B., Oates-Indruchova, L., Slany, W., 2019. Female teenagers in computer science education: Understanding stereotypes, negative impacts, and positive motivation. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1903/1903.01190.pdf.

Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2017. An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. Softw. Eng. 43 (1), 1–18. http://dx.doi.org/10.1109/TSE.2016.2584050.

Teague, J., 2002. Women in computing: What brings them to it, what keeps them in it? ACM SIGCSE Bull. 34 (2), 147–158.

Tissenbaum, M., Sheldon, J., Abelson, H., 2019. From computational thinking to computational action. Commun. ACM 62 (3), 34–36. http://dx.doi.org/10.1145/3265747.

Tómasdóttir, K.F., Aniche, M., v. Deursen, A., 2017. Why and how JavaScript developers use linters. In: 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 578–589. http://dx.doi.org/10.1109/ASE.2017.8115668, 30 Oct.-3 Nov. 2017.

Videnovik, M., Zdravevski, E., Lameski, P., Trajkovik, V., 2018. The BBC micro: bit in the international classroom: Learning experiences and first impressions. In: 17th International Conference on Information Technology Based Higher Education and Training (ITHET). IEEE, Olhao, Portugal, pp. 1–5.

Vitores, A., Gil-Juárez, A., 2016. The trouble with 'women in computing': a critical examination of the deployment of research on the gender gap in computer science. J. Gend. Stud. 25 (6), 666–680.

Voogt, J., Fisser, P., Good, J., Mishra, P., Yadav, A., 2015. Computational thinking in compulsory education: Towards an agenda for research and practice. Educ. Inf. Technol. 20 (4), 715–728.

Voogt, J., Roblin, N.P., 2012. A comparative analysis of international frameworks for 21st century competences: Implications for national curriculum policies. J. Curric. Stud. 44 (3), 299–321.
Wang, M.-T., Degol, J.L., 2017. Gender gap in science, technology, engineering, and mathematics (STEM): Current knowledge, implications for practice, policy, and future directions. Educ. Psychol. Rev. 29 (1), 119–140. http://dx.doi.org/10.1007/s10648-015-9355-x.
Webb, H., Rosson, M.B., 2013. Using scaffolded examples to teach computational thinking concepts. In: 44th ACM Technical Symposium on Computer Science Education. ACM, Denver, Colorado, USA, pp. 95–100. http://dx.doi.org/10.1145/2445196.2445227, 2445227.
Weintrop, D., Wilensky, U., 2018. How block-based, text-based, and hybrid block/text modalities shape novice programming practices. Int. J. Child-Comput. Interact. 17, 83–92. http://dx.doi.org/10.1016/j.ijcci.2018.04.005.
Weintrop, D., et al., 2016. Defining computational thinking for mathematics and science classrooms. J. Sci. Educ. Technol. 25 (1), 127–147. http://dx.doi.org/10.1007/s10956-015-9581-5.
Werner, L., Denner, J., Campe, S., Kawamoto, D.C., 2012. The fairy performance assessment: measuring computational thinking in middle school. In: Proceedings of the 43rd ACM technical symposium on Computer Science Education. pp. 215–220.
Wilcox, C., Lionelle, A., 2018. Quantifying the benefits of prior programming experience in an introductory computer science course. In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education. pp. 80–85.
Wing, J.M., 2006. Computational thinking. Commun. ACM 49 (3), 33–35.
Wing, J.M., 2014. Computational thinking benefits society. In: 40th Anniversary Blog of Social Issues in Computing, Vol. 2014. p. 26.
Wohl, B., Porter, B., Clinch, S., 2015. Teaching computer science to 5-7 year-olds: An initial study with Scratch, Cubelets and unplugged computing. In: Workshop in Primary and Secondary Computing Education. ACM, London, United Kingdom, pp. 55–60. http://dx.doi.org/10.1145/2818314.2818340, 2818340.
Yadav, A., Hong, H., Stephenson, C., 2016. Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. TechTrends 60 (6), 565–568. http://dx.doi.org/10.1007/s11528-016-0087-7.
Zeller, A., 2009. Why Programs Fail: A Guide to Systematic Debugging. Elsevier.
Zhang, L., Nouri, J., 2019. A systematic review of learning computational thinking through Scratch in K-9. Comput. Educ. 141, 103607. http://dx.doi.org/10.1016/j.compedu.2019.103607.
Zhong, B., Wang, Q., Chen, J., Li, Y., 2016. An exploration of three-dimensional integrated assessment for computational thinking. J. Educ. Comput. Res. 53 (4), 562–590.

Dr. Mojtaba Shahin is a Research Fellow at Monash University, Australia. His research interests reside in Software Architecture, Empirical Software Engineering, Human Factors in Software Engineering, and Qualitative Research.

Ms. Christabel Gonsalvez is a Senior Lecturer at Monash University, Australia. Her main research area and expertise reside in Work Integrated Learning, Industry Engagement for Education and Employability.

Professor Jon Whittle is the Director of CSIRO's Data61, Australia. He is a world-renowned expert in Value-based Software Engineering, industrial adoption of Model-Driven Development (MDD), and Human–Computer Interaction (HCI), with a particular interest in IT for social good.

Dr. Chunyang Chen is a Lecturer at Monash University, Australia. His research focuses on Mining Software Repositories, Automated Software Development, Deep Learning, and Human-Computer Interaction.

Dr. Li Li is an ARC DECRA Fellow and Senior Lecturer at Monash University, Australia. His research interests are in the areas of Android Security, Static Analysis, Machine Learning, Deep Learning, and Empirical Study.

Dr. Xin Xia is the Director of the Software Engineering Application Technology Lab at Huawei, China. Prior to joining Huawei, he was an ARC DECRA Fellow and a lecturer at Monash University, Australia. His current research focuses on data science for software engineering, i.e., mining and analysing rich data in software repositories to uncover interesting and actionable information.