



Volume 5, No: 2
December 2021
ISSN 2513-8359

International Journal of Computer Science Education In Schools

Editors

Dr Filiz Kalelioglu

Dr Yasemin Allsop

www.ijcses.org

International Journal of Computer Science Education in Schools

December 2021, Vol 5, No 2

DOI: 10.21585/ijcses.v5i2

Table of Contents

	Page
Hayley C. Leonard, Sue Sentance Culturally relevant and responsive pedagogy in computing: A Quick Scoping Review	3 – 13
Merve Yildiz, Hasan Karal A Computer Science Unplugged Activity: <i>CityMap</i>	14 – 27
Elizabeth Rowe, Jodi Asbell-Clarke, Mia Almeda, Santiago Gasca, Teon Edwards, Erin Bardar, Valerie Shute, and Matthew Ventura Interactive Assessments of CT (IACT): Digital Interactive Logic Puzzles to Assess Computational Thinking in Grades 3–8	28 - 73
Jonathan Sharman, Claudia Ziegler Acemyan, Philip Kortum, Dan Wallach Good examples help; bad tools hurt: Lessons for teaching computer security skills to undergraduates	74 - 92

Culturally relevant and responsive pedagogy in computing: A Quick Scoping Review

Hayley C. Leonard

Sue Sentance

*Raspberry Pi Computing Education Research Centre,
University of Cambridge, Cambridge, UK*

DOI:

Abstract

The underrepresentation of certain groups in computing has led to increasing efforts to develop computing curricula that is responsive and relevant to a more diverse group of learners. The current paper used a Quick Scoping Review methodology to identify research that has implemented and evaluated culturally responsive and relevant K-12 computing curricula, and to understand how they have been designed, the methods used for evaluation, and the factors affecting their success. In total, 12 papers were included in the review, and all were from a United States setting. Successes included changing learners' attitudes towards computing and increased learning gains. Key factors in the implementation of the curricula were teacher confidence and understanding of the socio-political context of computing, opportunities provided for collaboration and sharing knowledge and opinions, and allowing time for difficult discussions without oversimplifying the issues. The review identifies important lessons to be learned for educators around the world who are aiming to increase diversity in representation in computing in their schools.

Keywords: culturally relevant, culturally responsive computing, equity, curriculum

1. Introduction

Recently, increasing attention has been paid to the cultural relevance of computing curricula. In this paper, we aim to highlight key areas of research and practice and identify factors influencing the success of interventions focused on underrepresented ethnic and cultural groups in computing. We will begin by providing an overview of key theories underpinning the work on cultural responsiveness in teaching. We will then review research focusing on the design and evaluation of computing resources that have culture, equity, and social justice as their focus. Finally, we will consider the implications of the research for the future development of computing curricula.

1.1 Theoretical Background

1.1.1 Cultural capital and Critical Race Theory

Cultural capital is an important, and often neglected, consideration when developing curricula or resources in education. The term refers to internal aspects of individuals that they share with members of their families and their communities, such as language, knowledge, and belief systems, but also to external products of culture, such as artistic expression (Bourdieu, 1986). Yosso (2005) has argued that the culture of dominant groups is that which is most valued in society and becomes the 'norm' by which other cultures or groups are judged.

According to Yosso (2005; p.75), although educational institutions may aim to provide neutral settings for learning, in reality they tend to promote standards and topics valued by the dominant groups in society, often resulting in “deficit thinking”. This results in students from non-dominant, or minority, groups being viewed as problems to be fixed, and who need to learn the dominant group’s cultures and ways in order to succeed in life (Cabrera, 2019). Several authors have highlighted how Critical Race Theory can help to identify, examine and challenge these deficit models, drawing from law, history and social theory to bring attention to how race and racism can implicitly bias and impact educational practices (e.g., Ladson-Billings & Tate, 1995; Lynn & Dixson, 2013; Yosso, 2005).

For example, having a computer at home allows a young person to develop computing skills and vocabulary, which is cultural capital that is valued in the school environment. Other children may have developed skills which are of great value in their culture but are not those that are valued by the dominant groups (Yosso, 2005). Thus, children from minority backgrounds can often be labelled sub-standard or deficient in certain ‘basic’ skills that are based on norms of middle-class white children which have been assumed to be applicable to children from all backgrounds (Lachney, 2017). Understanding these implicit assumptions and being open to identifying and supporting minority groups’ cultural capital is of great importance in the education system. One step towards this goal is through the development of more culturally relevant and accessible teaching approaches and resources for young people from different backgrounds, on which we will focus for the rest of this paper.

1.1.2 Developing a theory addressing culture and equity in computing education

Theories of equitable teaching and learning practices which focus on cultural relevance and responsiveness have developed over several decades, leading to frameworks such as Culturally Relevant Pedagogy (Ladson-Billings, 1995), Culturally Responsive Teaching (Gay, 2000), and Culturally Sustaining Pedagogy (Paris, 2012; see Madkins et al., 2020, for an overview of these approaches). The key elements of these frameworks all focus on how teachers understand, respond to and use the cultural diversity within their communities to help all students achieve, build relationships, celebrate and sustain their cultures, and understand and challenge inequitable practices and belief systems that marginalise minority groups.

Building on these frameworks in education, Scott and colleagues (2013, 2015) developed Culturally Responsive Computing (CRC) theory to focus on equitable teaching approaches in computing specifically. Through an extracurricular club for girls from minority groups (COMPUGIRLS) that was held over a period of two years, the authors promoted three main tenets of an equitable teaching approach: *asset building*, *reflection*, and *connectedness*. To address these tenets, the teacher/mentors were encouraged to identify and integrate the girls’ technological and subject area knowledge into the curriculum, to support the girls in reflecting on their own knowledge and how it had developed, and to explicitly make links with their peers and communities, and wider cultural/socio-political issues (Scott & White, 2013). After implementing the COMPUGIRLS program with a range of different groups, Scott et al. (2015) revised the theory to incorporate more nuance, resulting in the following five principles:

- 1) All students are capable of digital innovation
- 2) The learning context supports transformational use of technology
- 3) Learning about one’s self along various intersecting sociocultural lines allows for technical innovation
- 4) Technology should be a vehicle by which students reflect and demonstrate understanding of their intersectional identities
- 5) Barometers for technological success should consider who creates, for whom, and to what ends, rather than who endures socially and culturally irrelevant curriculum.

Scott et al. (2015; pp. 420-421)

These theoretical principles move away from a deficit model of thinking, in which the focus of computing teaching for minority groups is decontextualised ‘basic’ skills that they are seen to lack, and allows students to express their identities and their culture through technology in a way that is meaningful to them and their communities (Scott et al., 2015). They provide authentic learning experiences, in which students experience teaching and learning that is both interesting and relevant to them (“personal authenticity”) and which reflects ways of working within computer science that will be of use in their future careers (“professional authenticity”); Means & Stephens, 2021, pp.19-20).

The theory also promotes a critical engagement with technology and social justice issues to allow students to innovate and create technological solutions to address issues that affect them (Madkins et al., 2020). It allows students who have a focus on communal goals (i.e., those that involve working with, or for the good of, others; Brinkman & Diekman, 2016) to identify computing as a means of addressing these goals and, therefore, to feel a sense of belonging within the subject. Given that students from minority groups may have a greater focus on communal goals than those from dominant groups (Lewis et al., 2019), it is of great importance that computing as a discipline is seen to align with these goals in order to encourage more diversity in those choosing the subject and career.

Although based on implementation in an extracurricular setting, the COMPUGIRLS program and CRC theory have important applications for teaching computing in both formal and non-formal learning environments. The next section provides an overview of a range of curricula that have been developed for use in both environments that are focused on culture and equity, and that incorporate many of the principles of CRC theory.

1.2 Teaching computing through the lens of culture and equity

Over the last 20 years, researchers and practitioners have increasingly aimed to develop computing curricula that are culturally responsive and relevant. Designing equitable and authentic learning experiences in computing requires a conscious effort to take into account characteristics of learners and their social environments, and to deal with topics that are relevant to a wide range of students.

Means and Stephens (2021) outline three key areas on which educators should focus when developing authentic computing experiences: the *learners* (understanding who they are and the experiences they bring to the computing classroom), the *community* (recognising and understanding the knowledge, opinions and experiences of both the local and the learning community, and building on the cultural wealth in these communities), and the *activities* (identifying both personally and professionally meaningful computing tasks, as well as considering meaningful learning outcomes and how they are valued).

Madkins et al. (2020) describe three overlapping but separate equitable teaching practices in computing: *promoting identity development* by understanding the learners but also by allowing them to express themselves through computing; *highlighting the personal and sociopolitical relevance of technology* by situating technology ideas within their local community and wider sociopolitical context and allowing learners to address issues that are meaningful to them; and *positioning learners as creative agents/change agents* by empowering them to use technology to innovate and solve problems with personal relevance.

Both sets of principles suggest that students should not only be given the opportunity to express their cultural knowledge and identities in computing tasks, but also to develop a sense of identity and belonging within computing as a discipline and a profession. This involves challenging stereotypes of who can be a computer scientist and what computing should be used to achieve in society, as well as understanding and addressing power relationships and biases within the community of learners. Learners should feel empowered to be creative and work with others to innovate and solve problems important to them, their communities, and wider society.

1.3 The current paper

The current paper aimed to identify key approaches to embedding culturally responsive and relevant pedagogy into computing curricula and to understand the design principles that were used in the development of the curricula. Specifically, we were interested in approaches that had been used in formal and non-formal learning spaces with K-12 students and had been evaluated in terms of student outcomes. We utilised a Quick Scoping Review (QSR) methodology (Collins et al., 2015) to allow us to synthesise the evidence available and provide an informed conclusion concerning the previous literature. Although less rigorous than a full systematic review or meta-analysis, the QSR method remains transparent and minimises bias while allowing the author to answer more open-ended questions about the evidence on a subject (Collins et al., 2015). For the current paper, we aimed to address the following questions:

- 1) Which design principles have been embedded into the curricula to promote cultural relevance, equity and justice in computing?
- 2) How has the success of the teaching approaches or curricula been evaluated?
- 3) Which factors positively or negatively influence the success of the teaching approaches or curricula?

2. Methodology

2.1 Initial search

Our investigation of the literature took place between January and March 2021. We began by searching for a recent review paper on culturally relevant pedagogy in computing through Google Scholar, using the term “culturally relevant computing review”. This identified a narrative review conducted by Morales-Chicas et al. (2019) which included 22 papers from JSTOR Arts and Sciences, Web of Science, and ERIC databases on three key themes that had been identified in a previous scoping exercise, and which were used in the systematic literature search: culturally responsive computing, ethnocomputing, and Culturally Situated Design Tools. Ethnocomputing relies on the use of relevant cultural artifacts and symbols, as opposed to Western ones that pervade most computing curricula (Tedre et al., 2006). Culturally situated design tools (CSDTs) are developed to this end, producing visual programming media reflecting different cultural practices and artifacts, and co-designed with the community for whom the practice or artifact is relevant (Eglash et al., 2006). The term ‘e-textiles’ had also been added to the search terms as these teaching activities often include elements of culturally relevant pedagogy, although they do not specifically adopt a culturally responsive approach (Morales-Chicas et al., 2019). Papers were included in the review if they were written in English in peer-reviewed journals between 1998-2018, and if they focused on K-12 education.

2.2 Initial review

We read the review paper by Morales-Chicas et al. (2019), along with the 22 papers included in the review. For our current purposes, we decided to focus on studies evaluating outcomes for K-12 students receiving culturally relevant and responsive curricula, in both formal and non-formal education settings. Evaluations included quantitative or qualitative measures of learning, attitudes, engagement, or other outcomes, and were gathered from both students and teachers/instructors. Papers must be written in English.

From the 22 papers, two position and two review papers were excluded, along with one paper focused on teacher professional development, and four that did not specifically aim to take a culturally responsive approach. Seven more papers that described culturally relevant approaches without full evaluation were also excluded, resulting in a total of 6 studies remaining in the current paper from the original systematic review.

2.3 Additional search, review, and analysis

Since the review had covered literature up to 2018, we conducted an additional search for papers using the same search terms as the original review between 2018 and March 2021. We added the search term “Exploring Computer Science” to identify papers evaluating the school-based curriculum developed for high school students using culturally relevant and equity-focused approaches (Goode, 2010). We replaced the Web of Science database, to which we did not have access, with the ACM Digital Library. From JSTOR Arts and Sciences, we retrieved 8 additional papers across the original search terms, but none of them met the inclusion criteria. In ERIC, a further 9 papers were retrieved, with one paper meeting our inclusion criteria. ACM Digital Library produced three papers for inclusion from 86 results. Two additional papers from 94 potential results using the “Exploring Computer Science” search term were also found across the three databases.

The final QSR therefore included 6 papers from the original systematic review by Morales-Chicas et al. (2019), and 6 additional papers published between 2018 and March 2021.

3. Results

The 12 studies emerging from the QSR are presented in Table 1 and are linked to key design principles recommended by Madkins et al. (2020) and Means and Stephens (2021).

Table 1

Studies evaluating culturally relevant and responsive computing curricula in formal and non-formal K-12 settings

Author and Year	Focus of curriculum	Participants	Key equitable / authentic design principles
<i>Reviewed by Morales-Chicas et al. (2019)</i>			
<i>CSDTs</i>			
Eglash et al. (2011)	6 days of lessons using websites for fractal design - standard vs. culturally situated design tools (CSDTs)	40 10th Grade computing students (US)	Identity development and expression
Eglash et al. (2013)	After school clubs over period of 4 years using CSDTs	81 Grade 1-6 students in after school club (US)	Building on community/cultural knowledge
Babbitt et al. (2015)	3 days of lessons using websites for mathematical concepts in Ghanaian Adrinka symbols - standard vs. culturally situated design tools (CSDTs)	19 7th-8th grade students (Ghana)	
<i>Vernacular culture</i>			
Scott & White (2013)	After school/summer clubs over a 2-year period focusing on <i>asset building, reflections, and connectedness</i>	41 13–18-year-old girls from low income backgrounds (US)	Identity development and expression Positioning students as creative agents/change agents
DiSalvo et al. (2014)	Out-of-school program incorporating video games testing and workshops on programming	30 14–18-year-old male students (US)	Highlighting personal/sociopolitical relevance of technology and challenging biases Providing professionally authentic computing experiences
Ashcraft et al. (2017)	3 out-of-school courses over 9 months focusing on digital storytelling, designing and programming educational video games, and design and program projects in virtual worlds	28 12–17-year-old girls from low income backgrounds (US)	
<i>Additional studies of curricula with a culture/equity focus since 2018</i>			
<i>CSDTs</i>			

Davis et al. (2019)	In-class lessons incorporating cultural art and designs into a standard Python programming unit of work, using CSDTs	33 high school students (US)	Identity development and expression Building on community/cultural knowledge
---------------------	--	------------------------------	---

Scratch curricula

Franklin et al. (2020)	In-class Scratch Encore curriculum over one year (pilot), representing three strands: multicultural, youth culture, and gaming.	271 5th-8th Grade students (US)	Identity development and expression Building on community/cultural knowledge
Yang et al. (2021)	11-week library-based club with Scratch-based curriculum, focusing on key computing concepts and practices, and culturally responsive pedagogy	30 8–10-year-olds (US); 2 case studies	

ECS curriculum

McGee et al. (2018)	In-class ECS curriculum over 1 year, using inquiry-based approaches, culturally relevant and culturally responsive pedagogy	906 high school students (US)	Identity development and expression Building on community/cultural knowledge
Ryoo (2019)	In-class ECS curriculum over 1 year, using inquiry-based approaches, culturally relevant and culturally responsive pedagogy	70 high school students (US)	Positioning students as creative agents/change agents Highlighting personal/socio-political relevance of technology and challenging biases
Qazi et al. (2020)	In-class ECS curriculum over 1 year, using inquiry-based approaches, culturally relevant and culturally responsive pedagogy	398 high school students	

US = United States

Beginning with the review by Morales-Chicas et al. (2019), the length of the curricula ranged from three days to two years, with the majority of courses being delivered outside of school or in elective programs and with relatively low numbers of young people. Three studies utilised CSDTs, with results indicating improvements in students' attitudes towards computing/science (Eglash et al., 2011; Eglash et al., 2013), and/or in knowledge after lessons (Babbitt et al., 2015; Eglash et al., 2011), in comparison to groups not receiving the cultural curricula. Interestingly, Eglash et al. (2013) compared a heritage culture and a vernacular culture approach and reported a slight preference for vernacular culture amongst students. The authors point out that preferences and

interests are highly likely to change at different points in young people's development, which suggests that using a range of different cultural touch points may therefore provide the best results. It is important to note that these quantitative evaluations of relatively short, focused interventions relied on very small sample sizes and further research will be required to better understand the impact of using CSDTs on a larger scale and in the longer term. Furthermore, more research is required to understand differences between formal and non-formal settings in terms of how these tools are integrated into the wider curriculum, how much time is allowed for their exploration and use, and their impact on computing-specific knowledge and skills.

Some studies used a more vernacular culture approach, using video games and digital storytelling to engage young people from underrepresented groups in computing. The COMPUGIRLS program (Ashcraft et al., 2017; Scott & White, 2013) incorporates three courses: digital storytelling, designing educational video games using Scratch, and designing projects in virtual worlds. Over the course of the programs, the authors' qualitative analyses reveal changes in the girls' understanding of their own identities and their roles within the community through computing, as well as the development of key computing skills. DiSalvo and colleagues (2014) drew on the popularity of video games to engage African American male students in computing through a program in which students became games testers and participated in computing workshops. The program was considered successful in improving interest and confidence in computing amongst the students, with 65 percent choosing to study computer science after high school. The non-formal settings of these programs, along with the engaging content, seem to provide a platform for young people who are typically underrepresented in computing to explore different aspects of the subject and understand the relevance of it to their lives and future careers. Trying to incorporate these types of approaches into the formal education system can be more of a challenge, although recent studies (outlined below) are attempting to do so.

For example, in the studies since 2018, Eglash and colleagues have continued to evaluate the use of CSDTs in formal settings (Davis et al., 2019). Curricula based around the Scratch programming environment have also been developed and tested in and outside of school (Franklin et al., 2020; Yang et al., 2021). Finally, studies evaluating the rollout of the Exploring Computer Science (ECS) curriculum in schools have recently been published (McGee et al., 2018; Ryoo, 2019; Qazi et al., 2020). We will discuss each of these sets of studies in turn.

Davis et al. (2019) aimed to incorporate cultural computing into a standard Python programming course in schools and reported significant increases in students' learning of computing concepts during this time. Content that had taken nine months to learn in the standard course was covered in only six months in the adapted course. However, there was very little change in attitudes towards computing or evidence of increased understanding of the relationships between culture and computing as a result of the adapted curriculum. This perhaps suggests that there needs to be a very careful balance between the cultural and computing content in a curriculum, especially in formal schooling, to have an effect on both learning and cultural awareness or identity. Teachers who are delivering the curriculum have to ensure that there are clear learning gains from an activity which can be evidenced through formal testing and may have less time available to promote exploration and discussion than researchers or instructors delivering the same content in non-formal settings. However, repeating the study across different schools, teachers and students will be important in understanding the interplay between these factors outside of the single class of high school students involved in this particular study.

A greater focus on the computing content of the learning experience is clear in the two Scratch-based curricula designed for younger students in the studies by Yang et al. (2021) and Franklin et al. (2020). Yang et al. incorporated a culturally responsive computing element into a standard 'Use-Modify-Create' framework for teaching computing (Lee et al., 2011) in a non-formal setting. They highlighted opportunities in both the 'Modify' and 'Create' stages to develop a sense of identity and belonging and to produce personally relevant and meaningful artifacts through computing. In-depth case studies of two participants in the 11-week program revealed indicators of increased belonging to the community of their computing club, as well as clear progression in their understanding and implementation of computing concepts. However, further research is required with a much larger sample and with a greater focus on cultural identity outside of computing to evaluate this program's success.

The Scratch Encore curriculum (Franklin et al., 2020), on the other hand, has been piloted with 271 5th-8th Grade students in a school environment over the course of a year. It consists of sequential modules covering intermediate computing concepts, and each module is offered across three different strands: Multicultural, Youth Culture, and Gaming (all of which cover the same computing content but with different themes). The strands draw on both heritage and vernacular culture, and educators can choose which is the most relevant for their learners. Like the previous study, it utilises the Use-Modify-Create framework, providing opportunities for

personalisation and expressing an individual's identity through the projects that are created. The pilot evaluation revealed that teachers found the course engaging for their students and the difficulty level appropriate. Opportunities for incorporating culture and students' own identities was demonstrated through the projects students created, and teachers' feedback on the relevance of different lessons. Further research is required to assess the learning gains of students, as well as changes in their attitudes and confidence, using this course over time. Direct comparisons of the curriculum in formal and non-formal settings will also provide some insight into the impact of the purpose of the activities, and the role of the instructor and their instruction approaches, on student outcomes.

Based entirely in formal education settings, three recent studies have evaluated the implementation of the ECS curriculum over five different states (McGee et al., 2018; Ryoo, 2019; Qazi et al., 2020). Student engagement and interest in CS were positively affected across all three studies, with Qazi et al. reporting students' perceived improvement specifically in creativity, problem solving, critical thinking and collaboration. These improvements were related to how engaged students felt in the curriculum, including how autonomous they felt in solving real-life problems. The importance of real-world problem-solving for student engagement was also evident in Ryoo's qualitative analyses, along with being able to demystify CS practices and careers, and demonstrations of how students' voices and perspectives were valued and could make a difference to others. As well as changes in attitudes and engagement in response to the ECS curriculum, McGee et al. revealed significant learning gains in computational thinking over the course of the year, irrespective of students' gender and race. However, it is to be noted that both students' end of course attitudes and the teachers' years of teaching ECS were significant predictors of these learning gains.

The impact of teachers' years of experience may be due to increasing familiarity with the course content and their pedagogical and concept knowledge but is also likely to be related to their developing understanding of the equity-focused principles underlying the curriculum and their ease in discussing complex and sensitive issues around race, bias and systemic barriers (Goode, Ivey, Johnson, Ryoo, & Ong, 2020a; Goode, Johnson, & Sundstrom, 2020b). With additional years of experience, teachers also develop a better understanding of their own school and district in terms of local needs and relevant issues, as well as educational policies and resources available. Providing a strong professional development program for teachers and supporting them as they deliver the curriculum and navigate their specific educational contexts is vital for the success of culturally relevant and responsive computing in the classroom. Future research focused on teacher voice will also be of great importance to ensure that educators receive the full range of support they require to implement the curriculum.

4. Discussion

We used a Quick Scoping Review (QSR) method to investigate computing curricula that have been developed to be culturally relevant and responsive to a diverse group of learners. Specifically, we aimed to answer the following questions: 1) Which design principles have been embedded into the curricula to promote cultural relevance, equity and justice in computing? 2) How has the success of the teaching approaches or curricula been evaluated? 3) Which factors positively or negatively influence the success of the teaching approaches or curricula? We will begin by considering the first question, before turning to the issues of evaluating success.

4.1 Design principles

Across the 12 studies included in the QSR, the most common elements of the design were a) identity development and expression, and b) building on community or cultural knowledge. These elements are central to all culturally responsive teaching, allowing students to see themselves and their communities within computing and recognising and celebrating different types of knowledge and understanding (Madkins et al., 2020). In particular, the interventions focused on Culturally Situated Design Tools (CSDTs) emphasise these elements.

From the earlier set of studies reviewed by Morales-Chicas et al. (2019), the out-of-school programs (Ashcraft et al., 2017; DiSalvo et al., 2014; Scott & White, 2013) tended to focus on more vernacular culture and provided more opportunities for professionally authentic computing experiences, aiming to improve the learners' understanding of different careers and types of roles within computing. They also provided opportunities to identify and try to address issues that were meaningful to the learners and their communities, thus highlighting the relevance of computing and encouraging learners to become agents of change. From these studies, it could perhaps be inferred that non-formal learning spaces are more able to focus on these types of projects, given that they are less constrained by the curriculum and other pressures of teaching within the formal schooling system.

However, the later studies found through the QSR revealed that opportunities to highlight the personal and sociopolitical relevance of computing, to position learners as active agents of change, and to engage in open-ended and problem-solving activities is indeed possible within a formal computing curriculum (McGee et al., 2018; Ryoo, 2019; Qazi et al., 2020). The ECS curriculum incorporates these elements of culturally responsive teaching throughout its different units. This suggests that balancing the teaching of computing concepts and taking a culturally responsive approach can be done in the classroom but requires intentional and sensitive design.

4.2 How are curricula evaluated, and which factors affect their success?

Focusing on our second two questions, their interdependence means it is preferable to explore them together. The main outcomes measured across all 12 studies included in the QSR were related to student attitudes towards computing, with most studies using a survey measure or in-depth qualitative analyses to evaluate attitude change. Eglash et al. (2013) identified that a vernacular approach was relatively more popular with young people than the heritage approach, highlighting the importance of thinking about learners through an intersectional lens that takes into account a range of characteristics, and not focusing solely on heritage or ethnic background. DiSalvo et al. (2014) reported improved confidence in computing after their games testing workshops. One of the key factors in the changing attitudes of the young people was how they viewed participating as games testers as “cool” and that they were “paid to play” (p. 302), allowing them to present their enjoyment and interest in computing to their peers or families with pride rather than embarrassment. Thus, the people around the learners are also vital to the success of more vernacular approaches to teaching computing.

The qualitative studies additionally explored themes of identity, feelings of belonging to computing as a discipline, and changes to learners’ understanding of computing concepts over time (Ashcraft et al., 2017; Ryoo, 2019; Yang et al., 2021). Yang et al. identified the importance of collaborative work in increasing feelings of belonging and promoting shared skills and knowledge. Open discussions concerning sociocultural issues surrounding computing were also key to the success of the programs, both through the teachers’ willingness and competence to discuss these issues and the students’ feelings of comfort and trust to voice their own opinions (Ashcraft et al., 2017; Ryoo et al., 2019). Ashcraft et al. note that it takes time to develop the trusting relationships needed for these sorts of discussions, which suggests that opportunities should be embedded throughout a curriculum and not be presented as a one-off lesson or only early on in a course. This may seem to lend itself more to a non-formal setting which can take place over longer periods of time than students often receive for computing in schools. However, the Scratch Encore and ECS curricula demonstrate that careful planning and consideration can ensure that both computing content and opportunities for cultural responsiveness are incorporated over the course of a school year.

Two studies used objective measures of learning gains, providing evidence that content was covered more quickly using the culturally responsive approach (Davis et al., 2019) and that learners improved their computing understanding and skills over the year of the ECS course (McGee et al., 2018). Both of these studies identified the importance of the teacher in implementing culturally responsive teaching: the quantitative analysis conducted by McGee et al. revealed that years of teaching computing was a key predictor in student learning gains, while Davis et al. discuss the reluctance of the teacher to engage with certain elements of the sociopolitical content of the curriculum and to allow learners to choose their own projects. This reluctance may have been associated with the lack of change in student attitudes towards computing, despite their increased speed of learning. Recent studies have highlighted how important professional development is within the ECS program in terms of fostering a critical approach to educational practices, computing pedagogy, and teachers’ own implicit biases (Goode et al., 2020a, 2020b). The teachers’ understanding of culturally responsive teaching, the acceptance of its principles, and their proficiency in implementing them in the classroom are therefore key factors in the success of a culturally responsive approach to teaching computing. However, it is also important to consider the contexts in which teachers are delivering the curriculum, including school and district policies concerning teaching, and the time and resources allocated to computing. Research considering these aspects of teacher constraints will be an important next step in understanding how best to deliver culturally relevant and responsive computing in formal education.

4.3 Conclusion and implications for practice

Our review of studies reveals a number of different approaches to incorporating culturally responsive teaching into computing, ranging from short, focused interventions with targeted underrepresented groups, to full curricula implemented in the computing classroom. Most studies used a quantitative approach to evaluate the success of the programs, either through surveys of changing attitudes or objective measures of learning gains.

Positive changes in these measures tended to be more evident for longer interventions in which culturally responsive teaching was fully embedded rather than competing with the computing concepts being taught. Nevertheless, further research is required with larger samples and longer-term interventions, in many cases, and using validated instruments to investigate the effect of culturally responsive computing on students' attitudes and their later subject and career choices.

The studies utilising in-depth qualitative analyses revealed more about how the learner experience affected their changing attitudes towards computing: being able to openly discuss difficult subjects, think about the sociopolitical context of computing through a complex, intersectional lens, and being able to collaborate and share knowledge and opinions were central to learners' improving attitudes towards computing. Further qualitative studies are required in the future to provide more of this rich and detailed data concerning how and why attitudes may be changing amongst learners following a culturally responsive computing approach.

Future developments of culturally responsive approaches to teaching computing should incorporate the successful elements outlined above into their curricula. Furthermore, developing teacher knowledge and confidence in culturally responsive approaches will be key to their successful implementation. Professional development opportunities should be created alongside curricula to empower teachers and to allow them to fully support their learners in becoming innovators and agents of change within computing. Further research should also be undertaken to better understand teacher voice and the constraints on, and implications of, delivering culturally responsive computing in the classroom. By taking this approach, it may be possible to improve the diversity of representation amongst students choosing to continue with computing as a subject and a career.

Acknowledgements

We acknowledge the support of ACM SIGCSE in this work, which was partly funded by the Special Projects programme.

References

- Ashcraft, C. Eger, E. K., & Scott, K. A. (2017). Becoming technosocial change agents: Intersectionality and culturally responsive pedagogies as vital resources for increasing girls' participation in computing. *Anthropology & Education Quarterly*, 48(3), 233-251. doi:10.1111/aeq.12197
- Babbitt, W., Lachney, M., Bulley, E., & Eglash, R. (2015). Adinkra mathematics: A study of ethnocomputing in Ghana. *REMIE Multidisciplinary Journal of Educational, Research*, 5(2), 110-135. doi:10.17583/remie.2015.1399
- Bourdieu, P. (1986). The forms of capital. In: Richardson, J. (Ed.). *Handbook of theory and research for the sociology of education*, 241-258. Westport, CT: Greenwood.
- Brinkman, B., & Diekman, A. (2016). Applying the communal goal congruity perspective to enhance diversity and inclusion in undergraduate computing degrees. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 102-107.
- Cabrera, N.L. (2019). Critical Race Theory v. deficit models. *Equity & Excellence in Education*, 52(1), 47-54. DOI: 10.1080/10665684.2019.1630342
- Collins, A.M., Coughlin, D., Miller, J., Kirk, S. (2015). *The Production of Quick Scoping Reviews and Rapid Evidence Assessments: A How to Guide*. Joint Water Evidence Group.
- Davis, J., Lachney, M., Zatz, Z., Babbitt, W., & Eglash, R. (2019). A cultural computing curriculum. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 1171-1175.
- DiSalvo, B., Guzdial, M., Bruckman, A., & McKlin, T. (2014). Saving face while geeking out: Video game testing as a justification for learning computer science. *Journal of the Learning Sciences*, 23(3), 272-315.
- Eglash, R., Bennett, A., O'Donnell, C., Jennings, S., & Cintorino, M. (2006). Culturally situated design tools: Ethnocomputing from field site to classroom. *American Anthropologist*, 108(2), 347-362. doi:10.1525/aa.2006.108.2.347aa.
- Eglash, R., Gilbert, E. J., Taylor, V., Geier, S. R. (2013) Culturally responsive computing in urban, after-school contexts: Two approaches. *Urban Education*, 48(5), 629-656. doi:10.1177/0042085913499211.
- Eglash, R., Krishnamoorthy, M., Sanchez, J., & Woodbridge, A. (2011). Fractal simulations of African design in pre-college computing education. *ACM Transactions on Computing Education*, 11(3), 1-14, doi:10.1145/2037276.2037281.

- Franklin, D., Weintrop, D., Palmer, J., Coenraad, M., Cobian, M., Beck, K., ... & Crenshaw, Z. (2020). Scratch Encore: The design and pilot of a culturally relevant intermediate Scratch curriculum. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 794-800.
- Gay, G. (2000). *Culturally responsive teaching: Theory, research, and practice*. Teachers College Press.
- Goode, J. (2010). Connecting K-16 curriculum & policy: Making computer science engaging, accessible, & hospitable for underrepresented students. In *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education*, 42(1), 22-26. doi: [10.1145/1734263.1734272](https://doi.org/10.1145/1734263.1734272).
- Goode, J., Ivey, A., Johnson, S.R., Ryoo, J.J., & Ong, C. (2020a). Rac(e)ing to computer science for all: how teachers talk and learn about equity in professional development. *Computer Science Education*, DOI: 10.1080/08993408.2020.1804772.
- Goode, J., Johnson, S.R., & Sundstrom, K. (2020b) Disrupting colorblind teacher education in computer science. *Professional Development in Education*, 46(2), 354-367, DOI: 10.1080/19415257.2018.1550102
- Lachney, M. (2017) Computational communities: African-American cultural capital in computer science education, *Computer Science Education*, 27, 175-196, DOI:10.1080/08993408.2018.1429062.
- Ladson-Billings, G. (1995). Toward a theory of culturally relevant pedagogy. *American Educational Research Journal*, 32(3), 465-491.
- Ladson-Billings, G., & Tate, W. F. (1995). Toward a Critical Race Theory of education. *Teachers College Record*, 97, 47-68.
- Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. and Werner, L., (2011). Computational thinking for youth in practice. *ACM Inroads*, 2(1), 32-37.
- Lewis, C., Bruno, P., Raygoza, J., & Wang, J. (2019, July). Alignment of goals and perceptions of computing predicts students' sense of belonging in computing. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, 11-19.
- Lynn, M., & Dixon, A. D. (Eds.). (2013). *Handbook of critical race theory in education*. New York, NY: Routledge.
- Madkins, T. C., Howard, N. R., & Freed, N. (2020). Engaging equity pedagogies in Computer Science learning environments. *Journal of Computer Science Integration*, 3(2), 1-27. <https://doi.org/10.26716/jcsi.2020.03.2.1>.
- McGee, S., McGee-Tekula, R., Duck, J., McGee, C., Dettori, L., Greenberg, R. I., ... & Brylow, D. (2018). Equal outcomes 4 all: A study of student learning in ECS. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 50-55.
- Means, B.M. & Stephens, A. (Eds.). (2021). *Cultivating interest and competencies in computing: Authentic experiences and design factors*. Washington, DC: The National Academies Press. <https://doi.org/10.17226/25912>.
- Morales-Chicas, J., Castillo, M., Bernal, I., Ramos, P., & Guzman, B. L. (2019). Computing with relevance and purpose: A review of culturally relevant education in computing. *International Journal of Multicultural Education*, 21(1), 125-155.
- Paris, D. (2012). Culturally sustaining pedagogy: A needed change in stance, terminology, and practice. *Educational Researcher*, 41(3), 93-97. <https://doi.org/10.3102/0013189x12441244>.
- Qazi, M. A., Gray, J., Shannon, D. M., Russell, M., & Thomas, M. (2020). A State-wide effort to provide access to authentic Computer Science education to underrepresented populations. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 241-246).
- Ryoo, J. J. (2019). Pedagogy that supports computer science for all. *ACM Transactions on Computing Education (TOCE)*, 19(4), 1-23.
- Scott, K. A., Sheridan, K. M., & Clark, K. (2015). Culturally responsive computing: A theory revisited. *Learning, Media and Technology*, 40, 412-436. <https://doi.org/10.1080/17439884.2014.924966>.
- Scott, K. A., & White, M. (2013). COMPUGIRLS' Standpoint: Culturally responsive computing and its effect on girls of color. *Urban Education*, 48, 657 – 681. <https://doi.org/10.1177/0042085913491219>.
- Tedre, M., Sutinen, E., Kahkonen, E., & Kommers, P. (2006). Ethnocomputing: ICT in cultural and social context. *Communications of the ACM*. 49(1), 126-130. DOI: 10.1145/1107458.1107466.
- Yang, H., Coddling, D., Mouza, C., & Pollock, L. (2021). Broadening Participation in Computing: Promoting Affective and Cognitive Learning in Informal Spaces. *TechTrends*, 65(2), 196-212.
- Yosso, T.J. (2005). Whose culture has capital? A critical race theory discussion of community cultural wealth. *Race, Ethnicity and Education*, 8(1), 69-91, DOI: 10.1080/1361332052000341006.

A Computer Science Unplugged Activity: *CityMap*

Merve Yıldız¹

Hasan Karal²

¹Karadeniz Technical University

²Trabzon University

DOI:

Abstract

The purpose of this study is to introduce a developed activity which is named *CityMap* and to present an example of the implementation and evaluation process of a computer science unplugged activity through this activity. The aim of this activity is to write algorithms of going from one place to another using step by step instructions. The rules are also to reach the destination with the shortest way and the least number of steps using correct instructions. Accordingly, a map and a worksheet was designed to implement the *CityMap* which is related to daily life and scenario based. For the evaluation, an answer key was prepared and scoring criteria were determined. Then, the case study was used as a research method and the process of writing algorithms using step by step instructions of students was examined by implementing the the activity with 15 sixth grade students. Both individual and group evaluation were made and for this process game components were used. The findings revealed that, in general, students could wrote algorithms step by step instructions for the tasks determined in the activity. In addition, during the implementation, it was observed that using of the gamification made the activity more enjoyable.

Keywords: computer science unplugged, writing algorithms using step by step instructions, computational thinking, gamification

1. Introduction

In recent years, the concept of computational thinking has come to the fore as computer science has become a multidisciplinary field and the teaching of programming has spread rapidly at the K-12 education. Computational thinking, which is described as one of the 21st century skills (Grover 2018; Tabesh 2017), is seen as a basic literacy skill for digital age learners (Wing 2006). When the literature is examined, the concept of computational thinking was first put forward by Papert (1980) as *computational ideas*. According to this, in the most general sense, Wing (2006) defined computational thinking as *problem solving, system design and understanding of human behavior by making use of the concepts of computer science*. In another study, Aho (2012) described computational thinking as a thinking process that involves in formulating problems so their solutions can be represented as computational steps and algorithms. On the other hand, the International Society for Technology in Education (ISTE) and Computer Science Teacher Association (CSTA) published an operational definition about computational thinking. According to this definition, computational thinking is a problem solving process that includes (but is not limited to) the following characteristics: -Formulating problems in a way that enables us to use a computer and other tools to help solve them, -Logically organising and analysing data, -Representing data through abstractions such as models and simulations, -Automating solutions through algorithmic thinking, -Identifying, analysing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources, -Generalising and transferring this problem solving process to a wide variety of problems (CSTA and ISTE 2011). Although there are different definitions of the concept of computational thinking, it is emphasized in the literature that computational thinking is mostly related to problem solving and algorithmic thinking (Şahiner

2017; Kalelioğlu et al. 2016; ISTE 2015; Bundy 2007). In this context, for this study, computational thinking is considered as writing algorithms using step by step instructions towards the solution of a problem.

In the globalizing world, it is crucial to gain computational thinking skills (Barr and Stephenson 2011). In the Horizon Report (2017), it is stated that computational thinking skill is one of the six subjects that should be integrated into education at K-12 level. Similarly, in the report published by the European Commission, justifications for integrating computational thinking into educational programs are presented, such as improving 21st century skills and strengthening employment opportunities (Bocconi et al. 2016). For this reason, computational thinking has become the focus of studies in the field of education. When the literature is examined, it is seen that there are different teaching methods for the improvement of computational thinking skill. One of the popular tools that play a role in the improvement of computational thinking skill is computer science (Grover 2018). It can be said that there are two different approaches in the teaching of computer science: plugged and unplugged (Brackmann et al. 2017). For plugged; coding and robotics workshops, programming course, various digital tools and games, and for unplugged; paper-pencil activities, coloring, puzzles and card games can be given as examples.

Unplugged activities are one of the approaches that provide the teaching of computer science-related knowledge and skills that have become popular over the past few years. This approach can be adapted to almost any environment, as it enables educators to carry out their educational activities without the need for electricity, computers, internet or similar technological tools (Şendurur 2019; Nishida et al. 2009). Researches have shown that computer science unplugged activities contribute to the acquisition of basic concepts related to computer science (Hermans and Aivaloglou 2017; Wohl et al. 2015; Taub et al. 2009), support improvement of computational thinking (Leifheit et al. 2018; Jagušt et al. 2018; Rodriguez 2015), provide an entertainment element (like a magic show) for the lesson (Bell and Vahrenhold 2018; Curzon 2014) and help to overcome obstacles such as misconception or negative attitude towards programming (Bell and Vahrenhold 2018; Şendurur 2018). Additionally, it is possible to come across studies emphasizing that the use of game components for teaching computer science is an effective source of motivation (Tsarava et al. 2017; Kotini and Tzelepi 2015; Nishida et al. 2009), increases class participation performance (Ibanez et al. 2014; Swacha and Baszuro 2013) and strengthens collaboration (Li et al. 2013). Also Voigt et al. (2010) state that creating a competition learning environment with computer science unplugged activities provides a more enjoyable learning process. From this point of view, *CityMap*, which is related to daily life, scenario based and includes gamification, was developed. Thus, it is aimed to present an example of the implementation and evaluation process of a computer science unplugged activity through this activity.

2. Literature Review

The CS Unplugged (2019) organization, founded by Tim Bell, Ian Witten and Michael Fellows, was the first to define unplugged activities. They state that the purpose of these activities, which they call computer science unplugged, is to provide an understanding of the philosophy behind computer science, considering the difficulties experienced in teaching programming. Although programming is perceived as a difficult and complex process, the underlying algorithmic structure is actually a phenomenon that exists in daily life. Each of the daily routine tasks is an algorithm and these algorithms (tasks) are performed after they are modeled in mental processes. At this point, it is stated that computer science unplugged activities act as a bridge that facilitates this mental process (Şendurur 2018). Hence, Bell and Vahrenhold (2018) argue that computer science unplugged activities are a fun and flexible pedagogical approach that does not require any technological tools and are based on learning by doing and experiencing.

In a computer science unplugged activity conducted with 160 middle school students without programming experience, a significant increase was determined between the pre-test and post-test scores of the students in terms of programming and computational thinking (Threekunprapa and Yasri 2020). Leifheit et al. (2018) discussed computational thinking as algorithmic concepts in their study with 3rd and 4th grade students. As a result of the short test they applied at the end of each computer science unplugged activity, they found that students' interest and motivation towards programming were high and they concluded that the game-based computer science unplugged activity approach was useful in teaching algorithmic computational thinking. Besides that Brackman et al. (2017), in their quasi-experimental study with 5th and 6th grade students, found that there was a significant difference between the experimental and control groups in the development of students' computational thinking skills and reached the conclusion that this difference provided an evidence about the effectiveness of computer science unplugged activities.

In another experimental study, computer science unplugged activities were applied to students before teaching block-based programming (Hermans and Aivaloglou 2017). It was found that students used the concepts they learned about programming more and their self-efficacy was higher. Similarly, Wohl et al. (2015) compared

Scratch, Cubelets, and unplugged activities in teaching computer science to children aged 5-7, and stated that computer science unplugged activities made more contribution to the understanding of the concept of algorithm than others. Taub et al. (2009), in their study with middle school students, affirmed that unplugged activities are more effective in understanding computer science. Besides these, Nishida et al. (2009), in their study with 10 university students in the programming course, concluded that computer science unplugged activities are a fun way to understand and learn concepts and provide high motivation. In a three year longitudinal study conducted with grades 4-6 students, Jiang and Wong (2018) aimed to examine the effects of plugged and unplugged activities on the development of problem solving skills associated with students' computational thinking skill and their motivation for coding. The researchers suggested that students learned programming concepts well, gained problem solving skills, and plugged and unplugged activities both provided intrinsic motivation.

In summary, the findings of the studies indicate that computer science unplugged activities are effective method in teaching computer science and support the improvement of computational thinking skill such as problem solving and algorithmic thinking, which are sought in individuals in the 21st century. At this point, it is expected that this study, which sets an example for the design and implementation of computer science unplugged activities, will help instructors who are the practitioners, in planning lessons.

3. Method

Case study is one of the qualitative research approaches used to examine one or more situations in depth (Creswell 2009). The focus of the case study is to define an event as it exists, to examine the situation in its real environment and to describe it in detail (Leymun et al. 2017). Although there are different classifications regarding the case study in the literature, Yin (2014) divided the single and multiple case studies into holistic and embedded designs in line with the analysis units. According to this classification, holistic single case study design is used where a single unit of analysis is involved and a single case is analyzed. For this reason, since the purpose of the activity is to examine the process of writing algorithms using step by step instructions of the students in the study group via the *CityMap*, the study was designed with a holistic single case study.

3.1. Participants

In line with the purpose of the study, convenient sampling method, which is one of the sampling methods for qualitative research, was used because of easily accessible and applicable (Patton 2014). The study group consists of 15 (9 girls, 6 boys) sixth grade students who attended the "Coding and Robotics Education" course.

3.2. The Role of The Researcher

One of the researchers, the first author, is also the instructor of the course. In the first two weeks of the course, different activities were carried out for problem solving and writing algorithms step by step. And then in the third week, the developed *CityMap* activity was applied. During the implementation, the researcher only provided guidance on how to do the activity and did not make any intervention. In addition, the researcher made an observation about the students' behavior.

3.3. Development of The *CityMap* Activity

CityMap was developed as a result of reviewing various projects and literature on computer science unplugged activities and taking as basis. Especially, the activities were written by Bell, Witten and Fellows (1998), on the csunplugged.org, in the Information Technologies and Software Curriculum (MEB 2018) and the model was developed by Nishida et al. (2009) were examined. Based on these examinations, an activity table for *CityMap* was created (Table 1).

Table 1. The Computer Science Unplugged Activity *CityMap*

Activity name:	CityMap
Aim of activity:	Writing algorithms of going from one place to another using step by step instructions (go x square, turn right, turn left).
Rules of activity:	The rule is, using the correct instructions, to reach the destination with the shortest way (minimum number of squares) and the least number of steps (minimum number of code lines).
Learning outcomes:	<ul style="list-style-type: none"> • Solves a problem using the given instructions. • Writes algorithms using step by step instructions.
Age/Grade:	5th and 6th grade (can be adapted for all age groups according to the complexity of the given map and tasks)
Individual / Group Work:	Both individual and group work

Required preliminary skills:	Basic literacy skill Knowing the basic directions
Required materials:	A map and a worksheet
Evaluation tools:	Preparation of answer key Determination of scoring criteria Designing badges* (*Optional: can be used as a performance indicator at the end of the evaluation)

The aim of this activity is to write algorithms of going from one place to another using step by step instructions (go x square, turn right, turn left). The rule is, by using the correct instructions, to reach the destination with *the shortest way (minimum number of squares)* and *the least number of steps (minimum number of code lines)*. The activity was constructed based on a scenario in order to create sense of story (Bell and Vahrenhold 2018; Kelleher et al. 2007). In order to establish the relationship between the activity and daily life, daily routines are taken as a basis according to the scenario. Five different jobs character, namely *teacher, police, doctor, taxi driver and fireman*, were determined in the scenario. Later, three different tasks to be done in one day routine were created for the jobs characters. First task: to go from home to workplace, Second task: to go from workplace to the task place, Third task: to return home from the task place.

For example; as shown in Table 2, according to the scenario, the *teacher* lives in no 17, and his/her workplace is the school. The *teacher's* task is to go to the museum with the students. He/she will return home after the museum. In this case, the students who chose the *teacher* character were requested to write algorithms using step by step instructions, the way from home to school for the first task, from school to museum for the second task, and from the museum to home for the third task.

Table 2. The Scenario of *CityMap*

Jobs Character	Home	Workplace	Task place	Task (in the scenario)
Teacher (T)	No 17	School	Museum	"Today is the day of sightseeing .. Let's go to the museum with the students"
Doctor (D)	No 13	Hospital	Beach	"There is a case of injury at the beach .. Emergency!"
Police (P)	No 23	Polis Station	Market-1	"Oops!! There is a thief in the market.. "
Taxi Driver (TD)	No 22	Taxi Station	No 11	"You are expected from no 11."
Fireman (F)	No 28	Fire Department	No 16	"There was a fire at no16 .. Run and run .."

3.4. Implementation Process of The *CityMap* Activity

Before the implementation of the *CityMap* activity, the researcher wrote the houses and the tasks determined for five jobs character and pasted them on the map. During the implementation process, the students were asked to form groups of five and the students in each group to choose one of the five jobs characters. Since the study group consists of 15 students, three groups was formed. Later, materials were distributed as a map to each group and a worksheet to each student, the rules and what to do in the activity was explained. Everyone wrote algorithms using step by step instructions for three tasks on the worksheet individually in line with the jobs characters they chose (Figure 1). At the end of the lesson, the worksheets were collected to be checked.



Figure 1. Implementation process of *CityMap*

3.5. Data Collection and Data Analysis

In order to collect data, a map and a worksheet designed by the researchers were used for the *CityMap* activity in the study (Figure 2). The ways on the map were designed square by square, like a grid. In accordance with the scenario, buildings and houses such as hospitals, schools, parks, markets, which may be in a city, were placed around the ways on the map. In order to avoid confusion, door numbers were determined for the houses. On the other hand, the worksheet is divided into three separate sections for writing the determined tasks and each section is divided into lines. Small square spaces were added to the edge of each line where the steps are written. Thus, it was easier for the students to see how many steps (code lines) they did in the task. In order to determine how many squares the task was completed, it was asked to write the total number of squares at the end of the worksheet.



Figure 2. Map and worksheet designed for *CityMap*

The data collected via the worksheet was checked with the answer key prepared by the researchers. The answer key was created separately for the three tasks determined for each jobs character. According to the map, there are many alternative ways to go from one place to another in order to fulfill the determined tasks. But there is only one way that satisfies the activity rules. As an example, solutions are given in Table 3 regarding the three tasks of a *teacher*.

Table 3. An Example of Answer Key

Task1	Task2	Task3
Home → School	School → Museum	Museum → Home
1. Get out of the home (start)	1. Get out of the school (start)	1. Get out of the museum (start)
2. Move forward 1 square	2. Move forward 1 square	2. Move forward 1 square
3. Turn left	3. Turn right	3. Turn right
4. Move forward 3 square	4. Move forward 5 square	4. Move forward 6 square
5. Turn left	5. Turn left	5. Turn left
6. Move forward 3 square	6. Move forward 11 square	6. Move forward 1 square
7. Turn right	7. Turn right	7. Turn left
8. Move forward 1 square	8. Move forward 11 square	8. Move forward 1 square
9. Turn left	9. Turn right	9. Turn left
10. Move forward 1 square	10. Get in the museum (finish)	10. Move forward 3 square
11. Turn right		11. Turn right
12. Move forward 17 square		12. Move forward 3 square
13. Turn left		13. Turn right
14. Move forward 11 square		14. Get in the home (finish)
15. Turn right		
16. Move forward 5 square		
17. Turn left		
18. Get in the school (finish)		
Number of squares: 42	Number of squares: 28	Number of squares: 15
Number of code lines: 18	Number of code lines: 10	Number of code lines: 14

While evaluating, it was first checked that the students wrote algorithms using step by step instructions correctly. If the steps for each task were written correctly and completely, 10 points were given. If it is empty or mostly

incomplete, no points were given. The errors were grouped under 5 categories: 1- no writing start/finish step, 2- mixing direction (turn right/left step), 3- lacking of one or a few steps, 4- counting number of squares incorrectly, 5- using incorrect instructions. If there are errors in the writing algorithms using step by step instructions, 2 points were deducted from 10 points for each category according to these five error categories. Then, *the shortest way* (minimum number of squares) and *the least number of steps* (minimum number of code lines), which is the rule of the activity, was controlled. In this case, students who write algorithms using step by step instructions in accordance with both rules were given 4 points, 2 points for each rule. If it was written in accordance with only one rule, 2 points were given.

In case studies, it is recommended to use more than one data collection tool in order to obtain rich data about the situation under study (Patton 2014; Yıldırım and Şimşek 2013). In this study, the observation technique was used in addition to the worksheet. The researcher, who was the course instructor, made observations in the study environment as a *participant observer* and kept unstructured field notes about students' behavior. The data obtained from the field notes were used to support the findings.

3.6. Gamification and Assessment Framework

Gamification can be defined as the use of game philosophy, game components and game design techniques out of the context of game theory to increase motivation and encourage problem solving (Werbach and Hunter 2012; Deterding et al. 2011; Zichermann and Cunningham 2011). Although teaching via games or gamification basically has the same structural factors, while teaching factors are integrated into games in teaching via games, game components are integrated into existing teaching factors in gamification (Çağlar and Kocadere 2015). In other words, there is no game in the gamification of a teaching environment. Wherein the subject to integrate game components such as star, badge, level, leaderboard with the teaching environment. Thus, gamification, which is based on games, yields similar to the effects of games (Huotari and Hamari 2012).

In the researches were emphasized that the game components, which are the source of external motivation, have positive effects such as providing fun in the learning environment (De-Marcos et al. 2014; Kocadere and Çağlar 2015), increasing motivation (Sillaots 2014; Kocadere and Çağlar 2015, Su and Cahang 2015), ensuring engagement with the environment and increasing academic success (Ibanez et al. 2014; Su and Cahang 2015; Hanus and Fox, 2015). Therefore, points, badges and leaderboard were used in the evaluation of the *CityMap* activity. For individual evaluation; the answers of all students were checked and after the scoring was completed, the students who chose each jobs character were compared among themselves. For example; the scores of the students coded as T-1, T-2, T-3 for *teacher* were listed and the student with the highest score was given the *teacher badge* (Figure 3). If the scores were equal, the total number of errors was checked. For group evaluation; the teams that got the most badges to their group were determined and written on the board as the leaders of the week.



Figure 3. Badges designed for the evaluation of the *CityMap*

4. Results

While giving information about the implementation process of the activity at the beginning of the lesson, the researcher stated that this activity would be like a competition and that they could win badges individually and be enrolled on the leaderboard as a group at the end of the evaluation. The researcher observed that almost all students were excited and motivated to do the activity. The worksheets filled in by the students were checked through the prepared answer key. When Table 4 is examined, it is seen that 6 students in the first task, 4 students in the second task, and 6 students in the third task wrote algorithms using step by step instructions correctly and completely. According to this finding, although the number of students who wrote algorithms using step by step instructions without any error seemed low, when the number of errors was examined, it was found that some students made almost no errors ($f = 1$) in some tasks. For instance; while the student with code D-1 did not make any error in the

first and second tasks, in the third task, he made only one error in the category of *counting number of squares incorrectly*. As a result of the observations, it is thought that this situation is caused by the excitement of the students about the activity.

Written algorithms using step by step instructions by one student in the first task, three students in the second task, and two students in the third task were determined that most of them were incomplete or the activity was empty. According to Table 4, the student with the code P-3 completed the first and third tasks without error. When his/her worksheet was examined in detail, it was seen that he/she wrote only a few lines of the second task but did not complete it. On the other side, the student with the code D-2 did not complete the second and third tasks, while the student with the code F-3 student did not complete all three tasks. It was observed that these students were uninterested towards the lesson.

When the errors made regarding the writing algorithms using step by step instructions were examined, it was found that the most errors were made in the categories of *lacking of one or a few steps* ($f = 31$) and *counting the number of squares incorrectly* ($f = 22$). It is thought that this situation may be due to the long distance between the two places given in the relevant task or many alternative routes. Because it was observed that the students try too many times on the map to find the best solution. This situation may have led to confusion when writing algorithms using step by step instructions. Another situation is that one map was given for a group. Although the map was large, it was observed that sometimes, it was difficult for students to work on a map with five students.

Other errors related to writing algorithms using step by step instructions are *using incorrect instructions* ($f = 17$), *mixing direction* ($f = 15$) and *no writing start/finish step* ($f = 2$). However, when the number of errors is examined, it is seen that almost all of the errors in *using incorrect instructions* belong to the student with the code of T-3 (Table 4). When the worksheet of this student was examined in detail, it was determined that the student used incorrect instructions such as turn 5 square top, go up, not the given instructions. It can be said that the student made these errors because of misunderstanding about the use of the instructions. Similarly, it is seen that the majority of the number of errors in the *mixing direction* category belongs to the student with the code D-3. Here, it is the situation that the student confuses his/her own direction by the direction of relative to the map.

When the completion of activity according to the rule of *the shortest way* is examined, the number of students who finds correct solution is 10 in the first task, 8 in the second task, and 12 in the third task. The number of students who completes their tasks with *the least number of steps* is 9 for the first task, 6 for the second task, and 11 for the third task. These findings can be interpreted as students took into account the rules and focused on the activity to get points.

In addition to this, both individual and group evaluations were made by using the gamification. For individual evaluation; each jobs character group was evaluated within itself. The students who wrote the algorithms using step by step instructions correctly by following the rule of the shortest way and the least number of steps were ranked according to their total scores. In references to Table 4, the *teacher badge* is T-1, the *doctor badge* is D-1, the *police badge* is P-2, the *taxi driver badge* is TD-3 and the *fireman badge* was earned by F-2. During the announcement of the students who received the badges, it was observed that the students were curious and excited. For group evaluation; the group / groups that earned the most badges were enrolled on the leaderboard. In this case, the first and second group earned two badges each, and the third group just one. The two groups, who were the leaders of the week by getting equal badges, were observed that have shared their pleasure by hugging each other.

Table 4. Scoring of Writing Algorithms Using Step By Step Instructions for The *CityMap*

	Task1								Task2								Task3								Total number of errors	Total score		
	Errors				Rules				writing algorithms using step by step instructions	Errors				Rules				writing algorithms using step by step instructions	Errors				Rules					
	no writing start/finish step	mixing direction (turn right/left error)	lacking of one or a few steps	counting number of errors incorrectly	using incorrect instructions	the shortest way	the least number of steps	Score		no writing start/finish step	mixing direction (turn right/left error)	lacking of one or a few steps	counting number of errors incorrectly	using incorrect instructions	the shortest way	the least number of steps	Score		no writing start/finish step	mixing direction (turn right/left error)	lacking of one or a few steps	counting number of errors incorrectly	using incorrect instructions	the shortest way			the least number of steps	Score
+10p	-2p	-2p	-2p	-2p	-2p	+2p	+2p	+10p	-2p	-2p	-2p	-2p	-2p	+2p	+2p	+10p	-2p	-2p	-2p	-2p	-2p	+2p	+2p					
T-1	√					-	-	10	√					+	+	14	√					+	+	14	0	38		
T-2	/			1		+	-	10	/		2	3		-	-	6	/		1			+	+	12	7	28		
T-3	/				7	-	-	8	/				5	-	-	8	/				4	-	-	8	16	24		
D-1	√					+	+	14	√					+	+	14	/			1		+	+	12	1	40		
D-2	/		5	5	3	-	-	6	X					-	-	0	X					-	-	0	8	6		
D-3	/		5	3	1	+	+	8	/		3	3		+	+	10	/		3	1		+	+	10	19	28		
P-1	/		1			+	+	12	/		1	3	1	+	-	6	√					+	+	14	6	32		
P-2	√					-	-	10	√					+	-	12	√					+	+	14	0	36		
P-3	√					+	+	14	X					-	-	0	√					+	+	14	0	28		
TD-1	√					+	+	14	/			1		-	-	8	/		1	1	2	+	-	6	5	28		
TD-2	√					+	+	14	/			3	1	+	+	10	/			3		+	+	12	7	36		
TD-3	/			1		+	+	12	/			1	2	+	+	10	√					+	+	14	4	36		
F-1	/		1		2	+	+	10	/		1	3	2	-	-	4	√					+	+	14	9	28		
F-2	/	1		1	1	+	+	6	√					+	+	14	/			1		+	+	12	5	32		
F-3	X					-	-	0	X					-	-	0	X					-	-	0	X	0		
According to categories total number of errors	1	6	11	8	8					0	5	15	10	5				1	4	5	4	4						

T: Teacher, D: Doctor, P: Police, TD: Taxi Driver, F: Fireman

-1: Group1, -2: Group2, -3: Group3

√: Correct and complete, /: Some little errors, X: Empty or mostly incomplete

5. Conclusion and Discussion

The rapid advancement of science and technology, changing needs and expectations have started to differentiate the characteristics and competencies sought in individuals, and recently some skills such as problem solving, critical thinking, and creativity have come to the fore. These characteristics, called 21st century skills, are considered important for social innovation and economic growth, as well as they reveal individual differences. In particular, many smart systems called industry 4.0 enable data exchanges and production technologies to be automated and produce higher quality, cheaper and faster. The creation of these automatic systems is possible with artificial intelligence, robotic systems and various software. Thus, programming has become the new communication language of the digital world. So, unlike other skills, this has brought to the agenda the computational thinking skill, the most important feature of which is the opportunity for people to cooperate with computers (Demir and Seferoğlu 2017). In this context, computational thinking has taken its place in the development plans of many countries in order that learners can have skills that can keep up with the times (Bocconi et al. 2016) and started to become the focus of the studies in the field of education.

Computational thinking is a basic literacy skill for everyone, not only for computer scientists (Wing 2006). The most effective way to improve this skill is seen as programming, that is, computer science (Lye and Koh 2014; Wing 2011). Because programming is the process of developing an algorithm that will serve to solve a problem and its implementation (Akçay and Çoklar 2016), and many findings in the literature supported that programming improves students' cognitive skills (Chao 2016; Gülbahar and Kalelioğlu 2014; Fessakis et al. 2013). However, when it comes to programming teaching, which is perceived as a complex and difficult process, it is indicated that unplugged activities are an alternative teaching method for computer science (Thies and Vahrenhold 2016). From this point of view, CityMap was developed and thus, it is aimed to present an example of the implementation and evaluation process of a computer science unplugged activity through this activity. In line with this purpose, CityMap activity applied to 15 sixth grade students and the process of writing algorithms using step by step instructions of students, which is considered as computational thinking skill, was examined.

When the process of writing algorithms using step by step of students was examined; it was determined that most errors were made in the categories of *lacking of one or a few steps* and *counting the number of squares incorrectly*. It is thought that this situation may be due to the long distance between the two places given in the relevant task or many alternative routes or the use of a map by five students. Other common errors are *using incorrect instructions* and *mixing direction*. Almost all of the errors in *using incorrect instructions* belonged to one student. This situation indicates that the student does not understand the instructions. It can be said that in future studies, it may be beneficial to present the instructions to the students in written form, such as supported by visual examples, as well as verbally expressing them.

Similarly, it was determined that the majority of the error of *mixing direction* belonged to another student. When the student's answers were examined in detail, it was understood that the student confused his/her own direction by the direction of relative to the map. In a similar implementation, it is thought that giving a map to each student will be a solution to this problem. Another result is that most of the students have reached the solution with the shortest way and the least number of steps, as required by the rules of the activity. This indicates that the students can understand the rules and use the instructions correctly.

In addition to these, while giving information about the implementation process of the activity at the beginning of the lesson, it was observed that almost all students were excited when it was stated that they could win badges individually and be enrolled on the leaderboard as a group at the end of the evaluation. This finding indicates that

the game components help to make the lesson more fun. Hence in the literature, it is asserted that the inclusion of gamification and entertainment elements in unplugged activities is an effective source of motivation (Bell and Vahrenhold 2018; Tsarava et al. 2017; Voigt et al. 2010). As a conclusion, in this study, it was presented an example of the implementation and evaluation of a computer science unplugged activity and some suggestions were made for the implementation of this activity. In future studies, the effects of these and similar activities on students' computer science achievements or computational thinking skill can be examined.

Acknowledgements

A part of this study was presented at 7th International Instructional Technologies and Teacher Education Symposium (ITTES), Antalya, Turkey, 30 October-1 December 2019.

Funding information

This study was supported by The Scientific and Technological Research Council of Turkey (Project No: TÜBİTAK - 118R034).

References

- Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, 55(7), 832-835.
- Akçay, A., & Çoklar, A. N. (2016). Bilişsel becerilerin gelişimine yönelik bir öneri: Programlama eğitimi. In A. İşman, H. F. Odabaşı and B. Akkoyunlu (Eds.), *Eğitim Teknolojileri Okumaları 2016* (pp. 121-139). Ankara, TOJET.
- 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?. *ACM Inroads*, 2(1), 48-54. <https://doi.org/10.1145/1929887.1929905>
- Bell, T., & Vahrenhold, J. (2018). CS unplugged - How is it used, and does it work?. In H. J. Böckenhauer, D. Komm, and U. W. (Eds.) *Adventures Between Lower Bounds and Higher Altitudes. Lecture Notes in Computer Science* (pp. 497-521). Springer, Cham. https://doi.org/10.1007/978-3-319-98355-4_29
- Bell, T. C., Witten, I. H., & Fellows, M. (1998). Computer Science Unplugged: Off-line activities and games for all ages. Retrieved April 25, 2020 from: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.114.7908&rep=rep1&type=pdf>
- Bocconi, S., Chiocciariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education - Implications for policy and practice*. Seville: Join Research Center (European Commission). <https://doi.org/10.2791/792158>. Retrieved May 15, 2020 from: https://publications.jrc.ec.europa.eu/repository/bitstream/JRC104188/jrc104188_computhinkreport.pdf
- 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 *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (pp. 65-72).
- Bundy, A. (2007). *Computational thinking is pervasive*. Retrieved May 23, 2020 from: <https://www.inf.ed.ac.uk/publications/online/1245.pdf>

- Chao, P. Y. (2016). Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers & Education, 95*, 202-215. <https://doi.org/10.1016/j.compedu.2016.01.010>
- Creswel, J. W. (2009). *Research design: Qualitative, quantitative, and mixed methods approaches*. Los Angeles, University of Nebraska-Lincoln.
- CS Unplugged (2019). *About*. Retrieved October 25, 2019 from: <https://csunplugged.org/en/about/>
- CSTA & ISTE (2011). *Operational definition of computational thinking for K-12 education*. Retrieved November 30, 2017 from: <http://www.iste.org/docs/ct-documents/computational-thinking-operational-definitionflyer.pdf>
- Curzon, P. (2014). *Unplugged computational thinking for fun*. Retrieved April 25, 2020 from: https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/8257/file/cid07_S15-27.pdf
- Çağlar, Ş., & Kocadere, S. A. (2015). Çevrimiçi öğrenme ortamlarında oyunlaştırma [Gamification in online learning environments]. *Journal of Educational Sciences & Practices, 14*(27), 83-102.
- De-Marcos, L., Domínguez, A., Saenz-de-Navarrete, J., & Pagés, C. (2014). An empirical study comparing gamification and social networking on e-learning. *Computers & Education, 75*, 82-91. <https://doi.org/10.1016/j.compedu.2014.01.012>
- Demir, Ö. & Seferoğlu, S. S. (2017). Yeni kavramlar, farklı kullanımlar: Bilgi-işlemsel düşünmeyle ilgili bir değerlendirme. In H. F. Odabaşı, B. Akkoyunlu, and A. İşman (Eds.). *Eğitim Teknolojileri Okumaları 2017* (pp. 468-483). Ankara, TOJET.
- Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From game design elements to gamefulness: "Defining gamification". In Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments (pp. 9-15).
- Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5-6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87-97. <https://doi.org/10.1016/j.compedu.2012.11.016>
- Grover, S. (2018). *The 5th 'C' of 21st century skills? Try computational thinking (not coding)*. Retrieved October 25, 2019 from: <https://www.edsurge.com/news/2018-02-25-the-5th-c-of-21st-century-skills-try-computational-thinking-not-coding>
- Gülbahar, Y., & Kalelioğlu, F. (2014). The effects of teaching programming via Scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education, 13*(1), 33-50.
- Hanus, M. D., & Fox, J. (2015). Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. *Computers & Education, 80*, 152-161. <https://doi.org/10.1016/j.compedu.2014.08.019>
- Hermans, F., & Aivaloglou, E. (2017). To Scratch or not to Scratch? A controlled experiment comparing plugged first and unplugged first programming lessons. In Proceedings of the 12th workshop in primary and secondary computing education (pp. 49-56).
- Horizon Report (2017). *The NMC/CoSN Horizon Report: 2017 K-12 Edition*. Retrieved November 13, 2019 from: <https://library.educause.edu/~media/files/library/2017/11/2017hrk12EN.pdf>

- Huotari, K., & Hamari, J. (2012). Defining gamification: a service marketing perspective. In *Proceeding of the 16th International Academic MindTrek Conference* (pp. 17-22).
- Ibanez, M. B., Di-Serio, A., & Delgado-Kloos, C. (2014). Gamification for engaging computer science students in learning activities: A case study. *IEEE Transactions on Learning Technologies*, 7(3), 291-301.
- ISTE (2015). *CT leadership toolkit*. Retrieved October 30, 2017 from: <http://www.iste.org/docs/ct-documents/ct-leadership-toolkit.pdf?sfvrsn=4>
- Jagušt, T., Krzic, A. S., Gledec, G., Grgić, M., & Bojic, I. (2018). Exploring different unplugged game-like activities for teaching computational thinking. In *2018 IEEE Frontiers in Education Conference (FIE)* (pp. 1-5). IEEE.
- Jiang, S., & Wong, G. K. (2018). Are children more motivated with plugged or unplugged approach to computational thinking?. In *Proceedings of the 49th ACM technical symposium on computer science education* (pp. 1094-1094).
- Kalelioğlu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic J. Modern Computing*, 4(3), 583-596.
- Kelleher, C., Pausch, R., Pausch, R., & Kiesler, S. (2007). Storytelling alike motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI Conference On Human Factors in Computing Systems* (pp. 1455–1464). ACM.
- Kocadere, S. A., & Çağlar, Ş. (2015). The design and implementation of a gamified assessment. *Journal of e-Learning and Knowledge Society*, 11(3), 85-99.
- Kotini, I., & Tzelepi, S. (2015). A gamification-based framework for developing learning activities of computational thinking. In *Gamification in Education and Business* (pp. 219-252). Springer, Cham.
- Leifheit, L., Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018). Programming unplugged: An evaluation of game-based methods for teaching computational thinking in primary school. In *12th European Conference on Game-Based Learning* (pp. 344)
- Leymun, Ş. O., Odabaşı, F., & Yurdakul, I. K. (2017). Eğitim ortamlarında durum çalışmasının önemi. *Eğitimde Nitel Araştırmalar Dergisi*, 5(3), 367-385.
- Li, C., Dong, Z., Untch, R. H., & Chasteen, M. (2013). Engaging computer science students through gamification in an online social network based collaborative learning environment. *International Journal of Information and Education Technology*, 3(1), 72-77.
- Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51-61. <https://doi.org/10.1016/j.chb.2014.09.012>
- MEB (2018). *Bilişim Teknolojileri ve Yazılım Dersi Öğretim Programı*. Retrieved October 25, 2019 from: <http://mufredat.meb.gov.tr/Dosyalar/2018124103559587-Bili%C5%9Fim%20Teknolojileri%20ve%20Yaz%C4%B1m%20-%206.%20S%C4%B1n%C4%B1flar.pdf>
- Nishida, T., Kanemune, S., Idosaka, Y., Namiki, M., Bell, T., & Kuno, Y. (2009). A CS unplugged design pattern. *ACM SIGCSE Bulletin*, 41(1), 231-235. <https://doi.org/10.1145/1508865.1508951>

- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, New York.
- Patton, M. Q. (2014). *Nitel araştırma ve değerlendirme yöntemleri*. [Qualitative research and evaluation methods]. (Trans. Eds. M. Bütün and S. B. Demir). Pegem Akademi, Ankara.
- Rodriguez, B. R. (2015). Assessing computational thinking in computer science unplugged activities. *Doctoral dissertation*, Colorado School of Mines, Arthur Lakes Library.
- Sillaots, M. (2014). Achieving flow through gamification: a study on re-designing research methods courses. In European Conference on Games Based Learning (vol. 2, pp. 538-545).
- Su, C. H., & Cheng, C. H. (2015). A mobile gamification learning system for improving the learning motivation and achievements. *Journal of Computer Assisted Learning*, 31(3), 268-286.
- Swacha, J., & Baszuro, P. (2013). Gamification-based e-learning platform for computer programming education. In X world conference on computers in education (pp. 122-130).
- Şahiner, (2017). Komputasyonel düşünme kavramı ile ilgili 2006-2016 yılları arasındaki bilimsel yayınların incelenmesi: Doküman analizi çalışması. *Yayınlanmamış Yüksek Lisans Tezi*, Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü.
- Şendurur, P. (2019). Investigation of pre-service computer science Teachers' CS-unplugged design practices. *Education and Information Technologies*, 24(6), 3823-3840. <https://doi.org/10.1007/s10639-019-09964-6>
- Şendurur, P. (2018). Programlama öğretiminde bilgisayarsız etkinlikler. In Y. Gülbahar and H. Karal (Eds.), *Kuramdan Uygulamaya Programlama Öğretimi* (pp. 189-235). Pegem Akademi, Ankara.
- Tabesh, Y. (2017). Computational thinking: A 21st century skill. *Olympiads in Informatics*, 11, 65-70. <https://doi.org/10.15388/loi.2017.special.10>
- Taub, R., Ben-Ari, M., & Armoni, M. (2009). The effect of CS unplugged on middle-school students' views of CS. *ACM SIGCSE Bulletin*, 41(3), 99-103.
- Thies, R., & Vahrenhold, J. (2016). Back to school: Computer science unplugged in the wild. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (pp. 118-123).
- Threekunprapa, A., & Yasri, P. (2020). Unplugged coding using flowblocks for promoting computational thinking and programming among secondary school students. *International Journal of Instruction*, 13(3), 207-222.
- Tsarava, K., Moeller, K., Pinkwart, N., Butz, M., Trautwein, U., & Ninaus, M. (2017). Training computational thinking: Game-based unplugged and plugged-in activities in primary school. In European Conference on Games Based Learning (pp. 687-695).
- Voigt, J., Bell, T., & Aspvall, B. (2010). Competition-style programming problems for computer science unplugged activities. In: Verdu, E., Lorenzo, R., Revilla, M., Regueras, L. (Eds.), *A New Learning Paradigm: Competition Supported by Technology* (pp. 207–234). CEDETEL, Boecillo.
- Yin, R.K. (2014). *Case study methods: design and methods* (5th ed.). Thousand Oaks: Sage Pbc.
- Werbach, K., & Hunter, D. (2012). *For the win: How game thinking can revolutionize your business*. Wharton Digital Press.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

- Wing, J. M. (2011). Research notebook: Computational thinking - What and why. *The link magazine*, 6. Retrieved January 18, 2020 from: <https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why>
- 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 *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 55-60).
- Yıldırım, A. & Şimşek, H. (2013). *Sosyal Bilimlerde Nitel Araştırma Yöntemleri* (9.baskı). Ankara, Seçkin Yayıncılık.
- Zichermann, G., & Cunningham, C. (2011). *Gamification by design: Implementing game mechanics in web and mobile apps*. O'Reilly Media, Inc..

Interactive Assessments of CT (IACT): Digital Interactive Logic Puzzles to Assess Computational Thinking in Grades 3–8

Rowe, E.¹ Asbell-Clarke, J.¹, Almeda, V.¹, Gasca, S.¹, Edwards, T.¹, Bardar, E.¹, Shute, V.², and
Ventura, M.³

¹*EdGE at TERC, Cambridge, MA, USA*

²*Educational Psychology and Learning Systems, Florida State University, Tallahassee, FL, USA*

³*Efficacy and Learning Research, Pearson, Boston, MA, USA*

DOI:

Abstract

The Inclusive Assessment of Computational Thinking (CT) designed for accessibility and learner variability was studied in over 50 classes in US schools (grades 3-8). The validation studies of IACT sampled thousands of students to establish IACT's construct and concurrent validity as well as test-retest reliability. IACT items for each CT practice were correlated to examine construct validity. The CT pre-measures were correlated with post-measures to examine test-retest reliability. The CT post-measures were correlated with external measures to examine concurrent validity. IACT studies showed moderate evidence of test-retest reliability and concurrent validity and low to moderate evidence of construct validity for an aggregated measure of CT, but weaker validity and reliability evidence for individual CT practices. These findings were similar for students with and without IEPs or 504s. IACT is the first CT tool for grades 3-8 that has been validated in a large-scale study among students with and without IEPs or 504s. While improvements are needed for stronger validity, it is a promising start.

Keywords: computational thinking, assessment, game-based learning, neurodiversity

1. Introduction

Computational Thinking (CT) has been attracting increased attention over the past decade in K–12 education, prompting a call for new models of pedagogy, instruction, and assessment (Shute, Sun, & Asbell-

Clarke, 2017; CSTA, 2017; National Academy of Sciences, 2010). The explosion of CT in education today parallels the turn towards scientific inquiry in science education in the 1990s (AAAS, 1993; Duschl, 1990; NRC, 1996). In 1996, the National Research Council issued new science education standards focusing on inquiry practices (NRC, 1996), yet even in 2018 there were few widely accepted tools for assessing students' scientific inquiry in classroom settings (Kruit, Oostdam, van den Berg, & Schuitema, 2018).

Developing learning assessments for any new focus of education is particularly challenging. In most educational research, new assessment methods are validated using existing "standard" measures of learning in the same content area. With an emerging field such as CT, no such standard measures exist. The few items that are in development and validation in today's research rely heavily on text and coding, which may preclude the measurement of CT for a broad range of learners. It is not only the novelty of the field that challenges the development of assessment in CT, it is also the nature of CT itself. Like scientific inquiry, CT is a thinking process. Measuring thinking processes is more nuanced than assessing whether or not a learner can solve a math problem or define a science term. Measuring learners' abilities to plan, design, and solve complex problems require methods for making thinking visible, which is not done by a typical school test (Ritchhart, Church, & Morrison, 2011). Even when CT is applied in a natural setting, such as in a coding environment, the final product may not reveal the CT practices as much as the thinking processes involved in designing code (Grover & Basu, 2017).

Addressing these issues for assessing CT may be particularly important to broadening participation in Computer Science and other Science, Technology, Engineering, and Mathematics (STEM) fields. Learning assessments often include irrelevant barriers (e.g., reading or coding prerequisites) that may mask conceptual understanding for some learners (Haladyna & Downing, 2004). Many learners who struggle academically because of neurodiverse conditions may have particular areas of strength in tasks related to CT, such as pattern recognition and systematic thinking (Baron-Cohen, Ashwin, Ashwin, Tavassoli & Chakrabarti, 2009; Dawson, Soulières, Gernsbacher, & Mottron, 2007; O'Leary, Rusch, & Guastello, 1991). Recognizing and nurturing these talents may be crucial for developing our future workforce (Martinuzzi & Krumay, 2013). In fact, many large IT companies, including Microsoft and Google, have programs specifically designed to recruit neurodiverse individuals (Wang, 2014). To capture this valuable expertise without the extraneous barriers that limit many neurodiverse learners' participation, a new form of learning assessment for CT is required.

This paper reports on the exploration of assessment items intended to measure CT within a game-based learning research study. Interactive Assessments of CT (IACT), designed for upper elementary- and middle-school students (grades 3–8), is a set of interactive, online, logic puzzles that were created to measure four fundamental CT practices: Problem Decomposition, Pattern Recognition, Abstraction, and Algorithm Design. The IACT assessment items were originally designed to be used as pre/post measures of CT practices in a study of the logic puzzle game *Zoombinis* (Asbell-Clarke, Rowe, Almeda, Edwards, Bardar, Gasca, Baker, & Scruggs, 2020). They were intended to identify evidence of CT Practices that: a) are apparent during the *process* of solving a task, as opposed to a final product; and b) are independent of a specific application and thus transferable or generalizable to other tasks. They were also designed to use as little text, specific coding notation, or other features that might impede some learners and/or mask their ability to solve CT problems. Because the

study specifically included learners who have Individual Education Plans (IEPs) related to academic struggles, the assessments needed to avoid extraneous factors that can impede some learners, such as the need to read and interpret complex word problems and excessive time pressure. The final constraint placed on the assessments was that they needed to be completed by all students within one class period (40-50 minutes, depending on the district). Students with IEPs were allowed up to 50 percent more time to complete the IACT assessments if necessary. In this paper, we report the findings from validation studies of IACT using two samples of elementary- and middle-school students, each with thousands of students, to understand IACT's construct and concurrent validity as well as test-retest reliability.

2. Background

The online logic puzzles that make up IACT were designed to serve as external pre/post assessments in a national, game-based learning study of over 50 upper elementary- and middle-school classes during a study of implicit CT practices demonstrated in *Zoombinis* gameplay in the 2017-18 school year (Asbell-Clarke, et al., 2020). Unfortunately, no validated instrument was available to measure CT practices at these grade levels when we started the study, so we designed two sets of IACT logic puzzles, one version for upper elementary and one version for middle school, each version with two comparable forms. During the *Zoombinis* study, we collected the pre/post IACT data along with teacher ratings of their students' CT at the end of the study. We used the teacher ratings to try to establish the concurrent validity of the IACT items. Because this method was not as rigorous as we would have liked, we extended the study to a second district-wide sample of over 3,000 students in grades 2–8 in a mid-sized Northeastern public school district. During the 2017-18 and 2018-19 school years, we were able to collect IACT data in May/June of each school year as well as corresponding items from another external instrument (Bebras) for the students in grades 5–8 in 2018-19. This paper reports on the findings of both of these samples for the validation studies of IACT.

2.1 Background on the Measurement of Computational Thinking

CT is a way of thinking used to design systematic and replicable ways to solve problems, emphasizing Abstraction and Algorithmic Thinking (Shute, Sun, & Asbell-Clarke, 2017; Wing, 2006). Rooted in ideas from research for the LOGO environment (Papert, 1980, 1991), CT includes practices and understandings dealing with logic, representations, and sequential thinking, as well as broader ways of thinking such as tolerance for ambiguity, persistence in problem solving, and abstraction across applications (Allan et al., 2010; Barr & Stephenson, 2011; Brennan & Resnick, 2012; Grover & Pea, 2013; Weintrop et al., 2016). Barr and Stephenson (2011) suggest that, in K–12, CT involves problem-solving skills and particular dispositions, such as confidence and persistence, when confronting particular problems. CT is also seen to be related to creativity and innovation (Mishra, Yadav, & the Deep-Play Research Group, 2013; Repenning et al., 2015) as well as integrating into many STEM areas (Barr & Stephenson, 2011; Sengupta, Kinnebrew, Basu, Biswas, & Clark, 2013; Weintrop et al., 2016).

In designing a middle-school curriculum called Foundations for Advancing Computational Thinking (FACT), Grover, Cooper and Pea (2014) used pedagogical strategies to support transfer from block-based to text-based programming, along with formative and summative assessments (including quizzes and tests as well

as open-ended programming assignments) related to the acquisition of computational thinking skills. Their findings show that students ages 11–14 using the FACT curriculum experience improved algorithmic learning, understanding of computing, and transfer of skills from the introductory programming environment, *Scratch*, to a text-based programming context. Building on this research, Lundh, Grover, Jackiw, and Basu (2018) suggest a framing of Variables, Expressions, Loops, and Algorithms (VELA) to prepare young learners for CT.

Many of the CT assessments developed to date are strongly tied to computer-science frameworks and rely on the construction or analysis of coding artifacts (Tang, Yin, Lin, Hadad, & Zhai, 2020). These include assessments such as the Fairy Assessment (Werner, Denner, Campe, & Kawamoto, 2012), Dr. Scratch (Moreno-León & Robles, 2015), Ninja Code Village (Ota, Morimoto, & Kato, 2016), REACT (Real Time Evaluation and Assessment of Computational Thinking) (Koh, Basawapatna, Nickerson, & Repenning, 2014), CodeMaster (von Wangenheim, et al., 2018) and tools developed by Grover, Cooper, and Pea (2014), which are all designed for specific programming environments like Alice, Scratch, AgentSheets, App Inventor, Snap!, or Blockly. As such, these tools may not be well-suited for use as pre-assessments or for use with interventions that are not primarily focused on coding (Wiebe, London, Aksit, Mott, Boyer, & Lester, 2019).

Recent initiatives to integrate CT with STEM require assessments that are more decontextualized or domain-general (Tang, et al., 2020; Karalar, & Alpaslan, 2021). The Computational Thinking test (CTt) (González, 2015) and Bebras Tasks (Dagienė & Futschek, 2008; Dagienė, Stupurienė, & Vinikienė, 2016) are two such instruments that have shown promise in assessing core CT constructs for middle-grades students (Wiebe et al., 2019). The CTt is an online, 28-item, multiple choice instrument shown to be valid and reliable with middle-school students in Spain (Román-González, Moreno-León, & Robles, 2017). Although designed for students with no programming experience, some items on the CTt have block-based, programming-like elements in them. However, research studies have not shown this to be problematic for students who reported having little or no prior programming experience (Wiebe et al., 2019). This result is supported by Weintrop, Killen, Munzar, and Franke (2019), who found that students perform better on questions presented in block-based form compared to text-based questions.

Bebras Tasks, which originated as a set of short competition tasks through which students in grades 5–12 apply CT to solve “real life” problems, have recently been looked at as assessment tools because their items map well to CT constructs (Barendsen et. al., 2015; Dagienė, Stupurienė, & Vinikienė, 2016; Izu, Mirolo, Settle, Manila, & Stupurienė, 2017). Like the CTt, Bebras Tasks do not rely on prior knowledge of an application or programming language, which makes them well-suited for use as a pre-assessment tool. The psychometric properties of Bebras Tasks have not been fully demonstrated and some tasks may be considered too peripheral to core CT skills (Román-González, Moreno-León, & Robles, 2017) for Bebras Tasks to stand alone as a standard assessment for CT in K–12 education. However, Wiebe and colleagues (2019) explored a promising hybrid assessment that includes items from both the CTt and Bebras as a “lean” assessment of current, generally recognized core CT skills. The Bebras items were most closely related to the intended constructs, grade band, and the nature of the logic puzzles that are the focus of this study, so we used selected Bebras items as external measures for evidence of concurrent validity for the logic puzzles with the second sample. The Bebras items are, however, more dependent on text than IACT and may present difficulties for students with certain IEPs.

2.2 Description of IACT Items

To measure foundational CT in grades 3–8, we developed a set of interactive logic puzzles focusing on four fundamental CT practices: Problem Decomposition, Pattern Recognition, Abstraction, and Algorithm Design. We chose to design a set of online puzzles because we were working with classes already using a web-based game, and the delivery and data collection for the assessment items could be integrated with the delivery of the game.

Knowing that these CT practices are rarely mutually exclusive within a set of activities, we identified a set of puzzles that might emphasize one practice over the others even if all practices were part of the activity. We designed the puzzles with minimal text and minimal prerequisite experience with coding or other specific activities. We drew inspiration from puzzle formats often used in psychological assessments, avoiding text and context-dependent scenarios. While these assessments are used in a variety of contexts involving executive functioning and reasoning; the overlap with CT practices is intriguing and merits study.

Theoretical Framing of Computational Thinking used in IACT

IACT was designed to measure the CT practices evident within the learning game *Zoombinis*. While not intended to include all potential facets of CT, IACT is grounded in emergent theoretical literature that is helping define the evolving constructs of CT in the educational field. The term CT was introduced by Jeanette Wing (2006) to describe the thought processes involved in formulating problems so that the solutions are represented in a form that can be effectively carried out by an information-processing agent (Cuny, Snyder, & Wing, 2010). The role of CT in K–12 education has been described as laying “the conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts” (Shute, Sun, & Asbell-Clarke, 2017). While many CT practices were discussed in Seymour Papert’s research on procedural thinking in the early programming environment for children called LOGO (Papert, 1980; Papert & Harel, 1991), today CT is thought to encompass much more than programming. There is also evidence that these CT practices may support a variety of other cognitive and non-cognitive activities, especially for learning in STEM subjects (e.g., Barr & Stephenson, 2011; Sneider, Stephenson, Schafer, & Flick, 2014).

Domain-general CT is often operationalized as a set of practices that include: problem decomposition, abstraction, algorithmic thinking, conditional logic, recursive thinking, and debugging (CSTA, Shute et al., 2017; Tang, et al., 2020). For the development of IACT, we focus on the CT practices that were most closely related conceptually to the puzzles in *Zoombinis* gameplay. We selected four fundamental CT practices outlined by CSTA (2017) and Shute, Sun, & Asbell-Clarke (2017):

- *Problem Decomposition* is reducing the complexity of a problem by breaking it into smaller, more manageable parts.
- *Pattern Recognition* is seeing trends and groupings in a collection of objects, tasks, or information.
- *Abstraction* is generalizing from observed patterns and making general rules or classifications about objects, tasks, or information by discerning relevant from irrelevant information.
- *Algorithm Design* is establishing reusable procedures that solve sets of problems.

While not an exclusive definition of CT, a focus on these practices lays a strong foundation for CT (CSTA, 2017). While CT can include many other practices such as modelling, debugging, and data visualization, this study focuses on these four CT practices because they are highly related to *Zoombinis* gameplay and they show promise of generalization to problem-solving in a variety of disciplines. When educating young learners in upper elementary and middle school, it may be important to ensure these broadly applicable practices have a solid foundation and upon which more nuanced facets of CT can be built.

2.3 Design of the IACT Items

The IACT items were designed for use in a game-based learning study where students may not have had any previous exposure to computer science or coding. While not designed as clinical assessments of executive functioning, the IACT items drew from models from similar psychological assessments that were designed for a broad range of neurodiverse learners to ensure maximum accessibility.

The authors worked with a game-based learning assessment company to design the IACT logic puzzles. Two sets of IACT items containing similar logic puzzle items were designed, one for upper elementary- and one for middle-school learners. The item sets were conceptually and structurally the same for both grade bands, but differed in terms of difficulty (e.g., based on the number of variables to consider in a pattern and size of the array for Abstraction problems). The item sets were distributed across two comparable forms for each grade band, a pre-test and a post-test, that were balanced and could serve as external pre/post measures of gains in our game-based learning studies. All items went through a minimum of two rounds of iteration and testing with think-aloud interviews with 8-10 students per round to test that the wording was eliciting the CT practices of interest.

The four fundamental CT practices that were evident in the *Zoombinis* gameplay (excerpted from Asbell-Clarke, et al., 2020), and thus formed the constructs measured with IACT are:

- **Problem Decomposition:** When approaching a complex problem, learners may need to simplify the problem—decomposing it into manageable parts and then tackling one part at a time. This is comparable to the practice of isolating variables in a science experiment or to factoring equations into terms in mathematics. Everyday examples of problem decomposition include taking the steps to bake a cake (choosing a recipe, gathering ingredients, mixing batter, baking, and frosting), or when planning a party (dealing with the guest list, then the menu, and then the music). When confronted with a multi-faceted puzzle (for example, sorting objects by both shape and colour), players often need to consider one part of the puzzle at a time (shape) and then consider the other (colour).
- **Pattern Recognition:** Pattern Recognition is required for all kinds of sorting and classification tasks as well as seeing trends in data and other forms of information. In science, learners look for patterns in the characteristics of animals to classify them into species, and patterns in the motions of planets to understand the laws of the universe. In math, learners use patterns to understand numbers and units, to collect like terms in an equation, and to generalize problems into classes of problems. Pattern Recognition is the precursor for abstraction, which is at the heart of CT.
- **Abstraction:** Abstraction is the ability to rise above the details and see the rules that can be applied generally to other situations. When learners can look across multiple instantiations of a phenomenon

and draw the common characteristics or patterns that can be abstracted, they are able to design generalized solutions to problems. For example, scientists are able to generalize the laws of gravity from the vast amount of observed evidence, and patterns within the evidence, that enable an abstracted claim about how the universe works. Similarly, mathematicians create systems of numbers and representations to exploit the inherent patterns of quantity in our world. The goal of abstraction is to design replicable systems of solutions that help us effectively and efficiently meet new challenges.

- **Algorithm Design:** Once abstracted, a set of rules can be operationalized through an algorithm. An algorithm is a sequence of instructions or steps required to accomplish a task. Everyday examples of algorithms include recipes in a cookbook, and consistent daily routines used to accomplish everyday tasks. Scientists design algorithms for replicable experimentation and for automated procedures required in large-scale data collection and analysis. Algorithms are used constantly in math ranging from standard processes of multiplication and division, all the way to abstract computer modelling of sophisticated phenomenon.

The example IACT items described in this section are from the elementary school test. A set of items for each CT practice was preceded by a warm-up item to familiarize students with the mechanics of the item.

Descriptions and illustrations of the test items are outlined in Table 1 and described in more detail below.

Table 1: Operationalization of CT Practices in IACT Assessment Items

CT Practice	Puzzle Type	Task	Measure
Problem Decomposition	Mastermind	Identify combination of colour and shape of item through testing	Efficiency: ratio of moves to required moves
Pattern Recognition	Raven's Progressive Matrix	Select one piece that best completes a pattern	Number correct
Abstraction	Sudoku	Fill grid spaces with coloured shapes according to a general rule	Percentage correct
Algorithm Design	Maze solving	Design a sequence of moves to complete the maze	Efficiency: ratio of moves to required moves

Problem Decomposition

Items related to problem decomposition involve a series of progressively harder puzzles that are similar to the game *Mastermind*. Students use feedback from the item to figure out which values (combination of colour and/or shape and/or pattern) solve the puzzle, The puzzle mechanic requires the student to drag objects to the test box to get feedback (i.e., correct or incorrect) with regard to colour and shape in as few moves as possible.

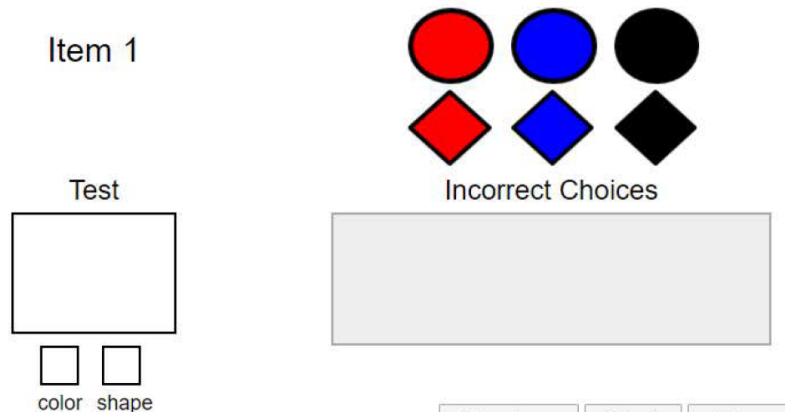
Figure 1 shows an easy problem. The correct answer for this item is “red diamond.” If the student first placed the

red circle in the test box, a green check will appear for “colour” and a red X would appear for shape. This tells the student to continue using a red object but not a circle. This leaves only the red diamond option, which is correct. The number of moves to solve the problem reflects the efficiency of problem decomposition skill.

Figure 1. Example logic puzzle item targeting Problem Decomposition

Drag one object at a time to the “Test” box to find the correct color and shape in as few tries as possible.

Place incorrect choices in the “Incorrect Choices” box.

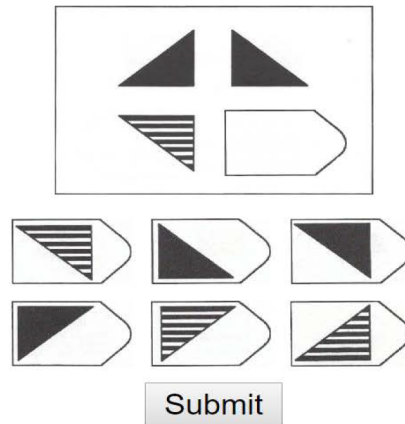


Pattern Recognition

Raven’s Progressive Matrices (RPM) (Raven, 1981) were used to assess Pattern Recognition. The RPM items serve as a baseline of learners’ ability to infer and apply different patterns in increasingly complex situations. RPM were designed to measure abstract reasoning involving patterns, and Raven (2000) pointed out that the RPM focuses on two components of general cognitive ability—making sense out of apparent chaos and generating a high-level schema to handle complexity. In the fairly easy item shown in Figure 2, the student needs to recognize that the top two black triangles are mirror images of each other, thus the bottom two should also be mirror images. Option 5 (middle item in the bottom row) is the correct response.

Figure 2. Example logic puzzle item targeting Pattern Recognition

Drag an option to the empty slot to complete the pattern and then hit the "Submit" button.

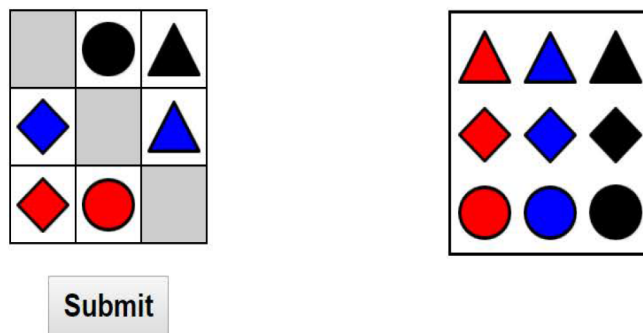


Abstraction

To assess Abstraction, pattern-matching puzzles were used. These puzzles require students to identify an underlying general rule associated with the patterns of objects and complete the puzzle by applying that rule. As shown in Figure 3, students drag objects from an inventory on the right into the grey cells on the left to complete the pattern, and thus applying the inferred rule. Each coloured shape can only appear once in the solution. In the example shown, the underlying pattern is relatively easy to discern—rows are the same colour, and columns are the same shape. Thus, the correct answer would be black square (upper left), blue circle (middle), and red triangle (lower right). In later tasks, the patterns are more complex and have more cells, thus generating more complex rules.

Figure 3. Example logic puzzle item targeting Abstraction

Drag objects from the box on the right into the gray cells to complete the pattern.
 Each colored shape can only appear once in your solution.
 Click "Submit" when you think you have it right.

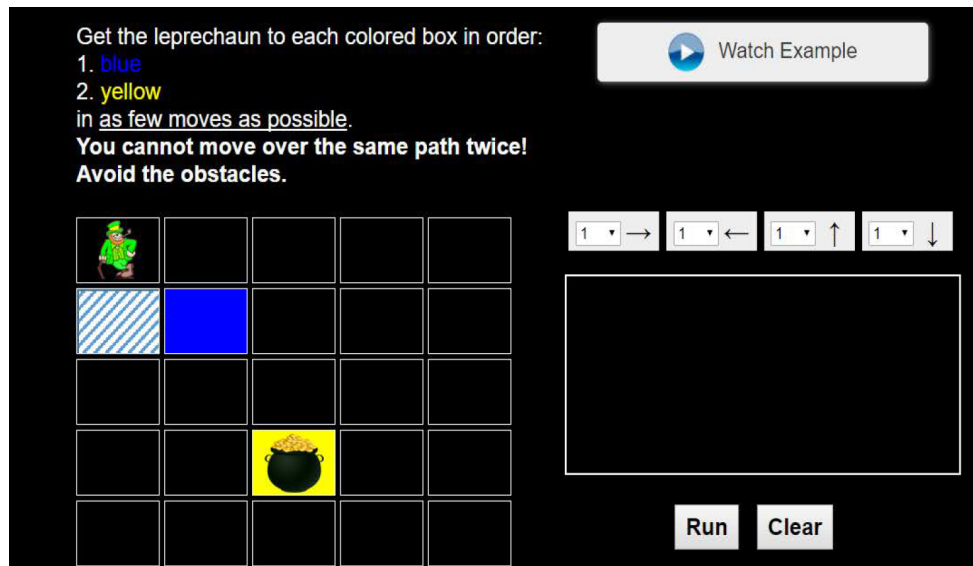


Algorithm Design

To assess Algorithm Design, the puzzles require a student to set up a sequence of arrows that will guide a character along a path in a maze that follows specified criteria. As shown in Figure 4, the sequencing task requires the student to insert directional arrows, along with the number of iterations needed to guide a leprechaun

to a pot of gold in the fewest steps possible, while avoiding certain obstacles. The fewest steps possible in this case is five (one step to the right, one step down, one step to the right, and two steps down).

Figure 4. Example logic puzzle item targeting Algorithm Design



These items were designed to measure individual CT practices as well as being aggregated for an overall CT measure.

The remainder of this paper discusses evidence for the validity and reliability of the IACT logic puzzles as measures of CT practices.

3. Research Questions & Hypotheses

The central question guiding this research is the extent to which the interactive logic puzzles provide a valid and reliable assessment of upper elementary- and middle-school learners' CT practices. To study this question, the results from students' performance on the puzzle tasks were examined using four types of analyses to test specific types of validity and reliability evidence with the following hypotheses about the results of each of these analyses:

- **Study 1:** Correlations among each set of IACT items (associated with each CT practice) were examined to establish evidence of construct validity—that all measures are distinct facets of the same broader CT construct.

Hypothesis 1: The CT practice measures will be moderately correlated with one another. Since the CT measures examine practices that are all facets of the same broader CT construct, we hypothesize that they will generally be aligned and share variance along the dimension of CT.

Study 2: The CT pre-measures were correlated with post-measures to establish evidence of IACT test-retest reliability.

Hypothesis 2: The pre-measures will be moderately to highly correlated with the post-measures, suggesting the versions of the logic puzzles are measuring the same construct at each time. The two sets of items designed were assumed to be equivalent and randomly assigned between the pre- and post-measures. It is intended that the variation between pre-and post-measures be attributable only to changes in the learners' practices rather than differences in the test questions themselves.

- **Study 3:** The CT post-measures were correlated with external measures (i.e., teacher ratings of their students' CT practices for one sample and students' scores on Bebras items for another sample) to examine concurrent validity.

Hypothesis 3: We hypothesize that the CT measures will be moderately correlated with teacher ratings of each CT skills and student scores of Bebras items, at the aggregate level and possibly at the level of each individual CT practice. The latter is questionable because of the amount of overlap among the individual practices.

- **Study 4: The validity and reliability analyses in studies 1-3 will be repeated separately for students with and without IEPs or 504 plan.**

Hypothesis 4: There will be no significant differences in the validity and reliability of IACT scores by IEP/504 status of the students.

3.1 Methods

The validation studies for IACT took place within other research studies. The first sample was collected during a national study of classes in grades 3-8 using the CT learning game, *Zoombinis*. This sample is referred to as the *Zoombinis* sample. The second sample was collected during a longitudinal study of the development of CT in grades 3-8 as part of our Research-Practice Partnership (RPP) with a mid-size suburban district outside a major Northeastern U.S. city. This sample is referred to as the RPP sample.

Zoombinis Sample

During the 2017-18 academic year, 146 teachers from 37 states and 6 countries applied to participate in the *Zoombinis* classroom implementation study. To participate, teachers needed to meet the following criteria:

- They are an elementary- or middle-school educator (grades 3-8) in the U.S.
- They teach at least one class that supports CT through logic, coding, or preparation for coding (e.g., math, science, computer science, tech ed., etc.).
- Their students have access to Internet-enabled computers to take the pre- and post- assessments required for the study.
- They complete a teacher agreement outlining the study requirements.
- They obtain administrative approval to participate in the study.

Forty-one teachers met all of these criteria and were accepted into the study. These teachers taught a total of 146 classes (10 charter, 21 private, 115 public). They were allowed to contribute a maximum of 3 unique classes in the study. To qualify as a unique class, the class must have either covered different subject areas or different grade levels of the same subject area. If teachers used *Zoombinis* in duplicate sections of these classes, their students took the pre- and post-assessments, but the teachers did not complete the other study data

collection requirements. These classes are considered “non-study” classes to avoid oversampling in the research studies, but they were retained for the validity and reliability analyses reported here. Fifty-seven of these classes were labelled as study classes (6 charter, 4 private, 47 public) and 91 were non-study classes (4 charter, 17 private, 70 public).

The initial *Zoombinis* student sample consisted of 3,234 elementary- and middle-school students across 146 classes from charter, private, and public schools (see Table 2). Of these, 2,456 students completed the pre-assessment and 1,828 completed the post-assessment. A total of 1,498 students completed both measures, belonging to 101 classes across 37 teachers. From the subset of 1,828 who completed the post-assessments, 851 students did not have a complete set of teacher ratings of their CT practices. These students were also excluded from the concurrent validity analyses, resulting in a sample size of 977 students with post-assessment and teacher ratings data.

RPP Sample.

Our second sample was collected from a small, suburban public-school district in the Northeastern U.S. as part of an RPP that has the mission to promote the infusion of CT into existing STEM curricula. As part of the longitudinal study on the impact of the RPP on students’ CT practices in 2017 - 2020, logic puzzles are administered to all students in grades 2–8 at the end of each school year. In grades 6–8, science teachers administered the logic puzzles during class time. In grades 2–5, technology teachers administered the logic puzzles during technology class time. All logic puzzles were completed in May-June of 2018 and 2019, because they are rising 6th graders 5th grade students took the middle-school forms of the test. For the RPP middle-school sample (grades 5–8), we also added five Bebras items to the pre- and post- assessments to provide evidence of concurrent validity.

The initial RPP student sample was comprised of 3,402 elementary- and middle-school students across the district in grades 2–8 (see Table 2) for 2017-18 and 3,697 students for 2018-19. Students who “never joined,” “withdrew” from the study, or did not have a complete set of pre-test and post-test scores were excluded from further analyses. The reduced samples across 6 elementary and 2 middle schools consisted of 3,066 students for 2017-18 and 2,909 students for 2018-19. A set of 1,414 students had complete pre-test and post-test scores during the first and second years of data collection---23% of these students had IEPs/504s ($N=337$), slightly above the national percentage of students with learning disabilities (Digest for Education Statistics, 2016; Horowitz, Rawe, Whittaker, 2017).

Table 2. Description of student samples

	<i>Zoombinis</i>			RPP		
	Pre-Assessment	Post Assessment	Both	2017-18	2018-19	Both
Total sample of students	3,234	3,126		3,402	3,697	

Never joined or withdrew from the study	262	154		146	300	
Incomplete Assessment Data	533	1155		178	476	
Final sample of students	2,439	1,817	1,435	3,078	2,921	2,301
Analyses	Construct Validity	Construct Validity	Test-Retest	Construct Validity	Construct Validity	Test-Retest

Data Cleaning

There were 2,788 *Zoombinis* students who either had pre-test or post-assessments. Two outliers from the performance on Algorithm Design (mean number of moves) were dropped. 18 students were excluded because they had completed an incorrect assessment for their grade level. This resulted in a total of 2,768 *Zoombinis* students. There were 3,666 RPP students who either had pre- or post-tests. Most of the students who had data from one year only were aging in (2nd grade in year 2) or aging out (8th grade in Year 1) of the sample. The final sample of RPP students with data from both years was 1,414. Details of data cleaning to arrive at the final sample of students who completed pre- and post-assessments can be found in Appendix A.

3.2 Data Collection

In each study, a variety of data was collected: student pre-post assessments, teacher logs of their CT instructional practices, and teacher interviews. In addition, as external measures of CT, teacher ratings of their students' CT practices were collected in the *Zoombinis* sample, and student scores on Bebras items were used for middle-school students (grades 5–8) in the RPP sample. The pre-post assessments and, when available, teacher CT ratings or student scores on Bebras items were used for the validation study of IACT assessments reported on in this paper.

Assessment Data Collection

In the IACT pre and post-tests, there were 3-6 items of increasing difficulty levels for each of the 4 practices of CT. All items had time limits for completion—2 minutes for the easier items and 5 minutes for the more difficult items. Teachers agreed to allot 30–45 minutes of class time for the administration of the pre-assessment and again for the post-assessment. Assessments were designed to take 30 minutes to allow students 150 percent of that to complete the assessment.

For the *Zoombinis* sample, teachers decided when to administer the pre-post assessments based on when *Zoombinis* best fit into their CT instruction. Teachers asked their students to complete the pre-assessments before they started playing *Zoombinis*. When teachers completed their CT instruction, they administered the post-assessment.

For the RPP sample, the IACT items were administered near the end of each school year through the district in grades 2–8. The data from Spring 2017 is used as the Time 1 measure for this study and data from Spring 2018 is used as the Time 2 measure.

All IACT data were collected through our team’s game data architecture, *Data Arcade*. *Data Arcade* facilitated the collection of all pre-assessment data and unlocked the game once the pre-assessment had been completed. Teachers created non-identifying usernames for their students in *Data Arcade*. Only teachers knew the real identities of students in their classes. *Data Arcade* then assigned a unique password and UserID number to each student. Teachers, in turn, shared the usernames and passwords with the students. This UserID was used to link assessment, game, CT rating, and Bebras data.

IACT Scoring

Pre-post assessment scale scores were calculated for the IACT logic puzzle items as the means of items per category: Problem Decomposition, Pattern Recognition, Abstraction, and Algorithm Design. This was calculated slightly differently for each set of tasks. The Problem Decomposition tasks provided feedback after each move, and the number of moves was unlimited, so the scores relied on the mean efficiency a student used to solve the puzzles. Efficiency is defined as the number of moves the student took divided by the minimum number of moves needed to solve the puzzle. In cases where the player lucked into getting a solution in less than the minimum number of moves, their efficiency was given a value of 1. The Pattern Recognition tasks simply had the student choose a response, so the scoring used the mean number of correct responses. Because the Abstraction puzzles allowed for individual array spaces to be counted as incorrect or correct, the mean percentage of spaces with correct responses was used. The Algorithm Design puzzles allowed for testing so the mean efficiency ($\# \text{ moves} / \text{minimum } \# \text{ moves needed}$) was also used in scoring. Because the first items for each category were used for practice, these items were dropped from mean calculations. Table 3 describes the calculations of the scale scores for each CT practice.

The middle school form was designed to be more difficult than the elementary form. To account for this, scores were standardized by form (elementary vs. middle school). The standardized scores for the CT practices were examined individually and in aggregate (Table 3). An aggregate measure of CT was calculated by first standardizing the means of each item type to produce a Z-score for each CT practice. The final Z-scores were averaged to create the aggregate CT measure used in this study. The units are the number of standard deviations from the mean Z-score of the four CT practices.

Table 3. Scoring of assessment items for each CT practice

CT Practice	Number of items	Measure used for scoring
Problem Decomposition	4	Mean efficiency ($\# \text{ moves} / \text{min } \# \text{ moves}$)
Pattern Recognition	5	Mean number of correct responses

Abstraction	6	Mean percentage of array spaces completed correctly
Algorithm Design	3	Mean efficiency (# moves/min # moves)
Aggregated CT	18	Average of Z-scores of 4 CT practice measures above

If a student had 0 number of moves on one of the Problem Decomposition or Algorithm Design items, this indicated that the student timed out of solving the puzzle. These timed-out instances were excluded from the computation of the mean efficiency in Problem Decomposition and Algorithm Design. Appendix B summarize the number of students who had complete, timed out, and missing pre-post assessments (mean efficiency) for Problem Decomposition and Algorithm Design, respectively.

Teacher CT Ratings Sheets

A CT rating sheet was designed and reviewed by 3-4 teachers before its use in the full study. Teachers in the study were given a brief description of the instrument by a research team member, along with its purpose and how to use it. After they administered the post-assessments, Zoombinis teachers were asked to rate each of their students based on the 4 CT practices in their students' work. This was an attempt to have an external measure of CT that still did not rely on text responses or coding. Sample behaviours were provided with a rubric so that teachers had a shared definition of each CT practice (See Table 4).

Table 4. Teacher ratings of their students' CT practices: Definitions and sample behaviours

CT Practice Definition	Sample Behaviours
Problem Decomposition: Breaking a problem into smaller, more manageable parts	1. When faced with a complex task, breaks it into smaller, simpler tasks.
	2. Considers one variable at a time when thinking about cause and effect.
Pattern Recognition: Identifying patterns, trends, or similarities between things	1. Identifies similarities and differences in sets of objects.
	2. Applies a pattern to predict an outcome.
Abstraction: Removing specific differences/details to make a generalized solution that will work for multiple problems	1. Identifies general rules to explain trends and patterns.
	2. Identifies common strategies that can apply to many problems
Algorithm Design: Creating an ordered series of instructions for solving a problem or performing a task	1. Writes or describes exact set of instructions for a complex task
	2. Recognizes the importance of the order of events in solving a problem.

Teachers received a Google Sheet with the *Data Arcade* usernames of students in each of their classes. Next to each student's username was a dropdown 5-point rating scale: Great Extent (5), Large Extent (4),

Moderate Extent (3), Slight Extent (2), Not at All (1). For each student, teachers were asked to select one rating for each CT practice based on the extent to which they had seen those behaviours in their students' work and overall classroom practices.

Bebras Items

Because the CT Rating sheet depended on teacher's ratings without substantive preparation or guidance, we also wanted to use a set of relatively established external items to compare results with the IACT items. We selected the Bebras tasks because they were closest to our needs, but we still had reservations that they would adequately measure CT practices among neurodiverse learners. We selected five items from Bebras that aligned with the four CT practices of our study. The first item was a maze task where students were to sequence a series of arrows to send a robot through a maze. This is analogous to the IACT items for Algorithm Design. The second item was a pattern matching game where shapes were combined to make another shape, analogous to the Raven's Progressive Matrices we used in IACT to measure Pattern Recognition. The third and fourth items were Problem Decomposition items analogous to the IACT Problem Decomposition item that mimicked the game *Mastermind*, but with considerably more text. The fifth Bebras item required students to generalize a rule to break a code, similar to the IACT Abstraction items. The five Bebras items used in this study are provided in Appendix C.

IEP/504 Plan Status

The RPP school district provided IEP/504 plan status for all students in their district each year of the study. Only students with IEP or 504 status of 'Active' in a specific school year were categorized as having an IEP/504 plan.

4. Results

To study the construct validity and reliability of the IACT items, addressing the first and second hypotheses, a series of correlational analyses were conducted using Pearson correlations with the following items: 1) pre-test measures, 2) post-test measures, and 3) test-retest of the same CT practice. To address the third hypothesis, namely concurrent validity, Pearson correlations were computed between post-test measures and teacher ratings of their students' CT for the *Zoombinis* sample, and between post-test measures and Bebras items for the RPP sample. All of these analyses were completed separately for students with and without IEP/504 plans to address the fourth hypothesis. Across all findings, mean moves was expected to be negatively correlated with mean number of correct responses and mean percentage of correct responses, as higher mean moves suggested less efficient solutions in the pre-test and post-test measures.

Construct Validity

Tables 5 and 6 display the correlations among the standardized IACT measures for the *Zoombinis* and RPP samples. In both samples, there are low to moderate correlations among the measures for both the pre- and the post-assessments (see Appendix D for the complete list of correlations). This supports the first hypothesis that the measures of the CT practices are similar yet distinct. The intercorrelation is highest between Pattern

Recognition and Abstraction, which points to the strong dependence of these practices. Abstraction can be thought of as the generalization of observed patterns into a rule or category, so it is reasonable that students who are strong in Abstraction would also be strong in Pattern Recognition.

Table 5. Pearson intercorrelations of pre-assessment measures for the *Zoombinis* and RPP samples

Correlations between CT Practice	CT Practice	<i>Zoombinis</i> sample (N= 2206-2314)	RPP sample (N= 2732-2937)	Average across samples
Problem Decomposition Avg Efficiency)	Pattern Recognition (Correct)	0.18	0.12	0.15
	Abstraction (Percent Correct Spaces)	0.23	0.16	0.20
	Algorithm Design (Avg Efficiency)	0.27	0.14	0.21
Pattern Recognition (Correct)	Abstraction (Percent Correct)	0.32	0.30	0.31
	Algorithm Design (Avg Efficiency)	0.24	0.21	0.23
Abstraction (Percent Correct Spaces)	Algorithm Design (Avg Efficiency)	0.26	0.26	0.26

Note: Significant at an alpha level of 0.0001.

Table 6. Pearson intercorrelations of post-assessment measures for *Zoombinis* and RPP samples

Correlations between CT Practices	CT Practice	<i>Zoombinis</i> sample (N= 1600-1773)	RPP sample (N= 2315-2623)	Average across samples
--	--------------------	---	--------------------------------------	-------------------------------

Problem Decomposition (Avg Efficiency)	Pattern Recognition (Correct)	0.19	0.12	0.16
	Abstraction (Percent Correct)	0.24	0.20	0.22
	Algorithm Design (Avg Efficiency)	0.22	0.14	0.18
Pattern Recognition (% Correct)	Abstraction (Percent Correct)	0.35	0.31	0.33
	Algorithm Design (Avg Efficiency)	0.23	0.23	0.23
Abstraction (Percent Correct Spaces)	Algorithm Design (Avg Efficiency)	0.24	0.23	0.24

Note: Significant at an alpha level of 0.0001.

Test-Retest Reliability

Table 7 displays the correlations for test-retest reliability among the standardized CT measures. Correlation coefficients may be higher for the *Zoombinis* sample than the RPP sample because all students in *Zoombinis* classrooms experienced some degree of CT intervention whereas this was true for less than a third of the RPP sample. There were varied results for the measures of individual CT practices, with acceptable test-reliability for the aggregated CT measure but below what was expected for the individual practices across both samples. The Pattern Recognition items were Raven's Progressive Matrices drawn from a public sample on the Internet. While there are no published test-retest results for these particular sets of items, this research typically has test-retest reliability of between 0.70 and 0.85 (e.g., Abdel-Khalek, 2005; Raven, Raven, & Court, 2000).

Table 7. Results for test-retest reliability

	<i>Zoombinis</i> sample (N= 1330-1434)	RPP sample (N=1955-2299)	Average across samples
Correlation coefficients for test-retest reliability	Pearson <i>r</i>		
Problem Decomposition (Avg Efficiency)	0.26	0.23	0.25
Pattern Recognition (% Correct)	0.21	0.14	0.18
Abstraction (% Correct Spaces)	0.38	0.41	0.40
Algorithm Design (Avg Efficiency)	0.27	0.18	0.23
Aggregated CT	0.55	0.43	0.49

Note: Significant at an alpha level of 0.0001.

While results for the measures of individual CT practices are considerably lower than what is indicated in prior literature, the finding for the aggregated CT measure in the *Zoombinis* sample indicate strong test-retest reliability. The aggregated CT measure is more stable across time as compared to measures of individual CT practices.

Concurrent Validity

Standardized measures of individual CT practices from IACT did not strongly correlate with the individual practices measured by the teacher ratings and scores on Bebras items (see Appendix E for results from the *Zoombinis* sample and the RPP samples, respectively). The correlations between the IACT measures and the external measures were no higher for corresponding practices than they were for non-corresponding practices. Neither teacher ratings nor student performance on comparable Bebras items were able to distinguish well between the individual practices of CT. This may likely be due to the highly overlapping nature of the CT practices discussed earlier. In the *Zoombinis* sample, the correlations between teacher CT ratings were moderately high, ranging from 0.70 to 0.78 for *Zoombinis* (see Appendix F), suggesting that these teachers were not distinguishing between CT practices when rating their students. There was some distinction between practices when using the Bebras items, however, suggesting that it may have been a limitation of the teacher rating sheet in supporting teachers’ distinction of the individual CT practices.

As seen in Table 8 while the individual practices were not correlated with the external measures, the aggregated measure of CT was moderately correlated with the teacher CT ratings for the *Zoombinis* sample, $r(941) = 0.29, p < 0.0001$ and with students’ Bebras scores for the RPP sample, $r(1408) = 0.40, p < 0.0001$. In particular, the IACT aggregated measure of CT was positively associated with an aggregated CT measure using five Bebras items, providing some evidence that these measures assess the same construct. In other words, students who performed better in all four CT practices as measured by IACT were also more likely to answer a higher percentage of the Bebras items correctly. While the correlations were moderate between aggregated measures of CT, the hypothesized moderate relationship between IACT and Bebras items at the individual CT practice level was not found. Those results can be found in Appendix E.

Table 8. Results for concurrent validity

Correlations between External CT measures	<i>Zoombinis Teacher Ratings (N=941)</i>	RPP sample (N=1408)	Average across samples
Aggregated CT	0.29	0.40	0.35

Comparison of Reliability and Validity by students with and without IEP/504s

All three analyses above included students with and without IEP/504s. In this section those analyses are repeated separately for students with and without IEP/504s and compared using a Fisher's Z-transformation in order to test the significance of differences between correlations from both groups.

Construct Validity

Tables 9 and 10 display the correlations among the standardized pre and post IACT measures for students with and without IEP/504s in the RPP sample. In both samples, there are low to moderate correlations among the measures for both the pre- and the post-assessments. After transforming these correlations to Fisher's Z scores, there were no significant differences in the construct validity by IEP/504 status of the students. This supports the fourth hypothesis that the measures of the CT practices are similar yet distinct regardless of student IEP/504 status.

Table 9. Pearson intercorrelations of pre-assessment measures for students with and without IEP/504s in the RPP samples

Correlations between CT Practice	CT Practice	<i>Students with IEP/504s</i> (N= 584-658)	<i>Students without IEP/504s</i> (N= 2230-2279)
Problem Decomposition (Avg Efficiency)	Pattern Recognition (Correct)	0.15	0.08
	Abstraction (Percent Correct Spaces)	0.16	0.12
	Algorithm Design (Avg Efficiency)	0.17	0.10
Pattern Recognition (Correct)	Abstraction (Percent Correct)	0.29	0.28
	Algorithm Design (Avg Efficiency)	0.16	0.21
Abstraction (Percent Correct Spaces)	Algorithm Design (Avg Efficiency)	0.27	0.23

Note: No differences between correlations were significant at an alpha level of 0.05.

Table 10. Pearson intercorrelations of post-assessment measures for students with and without IEP/504s in the RPP samples

Correlations between CT Practice	CT Practice	Students with IEP/504s (N= 462-573)	Students without IEP/504s (N= 1853-2049)
Problem Decomposition Avg Efficiency)	Pattern Recognition (Correct)	0.10	0.09
	Abstraction (Percent Correct Spaces)	0.21	0.16
	Algorithm Design (Avg Efficiency)	0.19	0.10
Pattern Recognition (Correct)	Abstraction (Percent Correct)	0.32	0.28
	Algorithm Design (Avg Efficiency)	0.29	0.20
Abstraction (Percent Correct Spaces)	Algorithm Design (Avg Efficiency)	0.30	0.19

Note: No differences between correlations were significant at an alpha level of 0.05

Test-Retest Reliability

Table 11 displays the correlations for test-retest reliability of the standardized CT measures among students with and without IEP/504 plans. There was no significant difference in test-retest reliability across these groups.

Table 11. Results for test-retest reliability by student IEP/504 status

Pearson r for test-retest reliability	Students with IEP/504s (N= 337)	Students without IEP/504s (N=1077)
Aggregated CT	0.41	0.37

Note: No differences between correlations were significant at an alpha level of 0.05.

Concurrent Validity

The aggregated measure of CT was moderately correlated with students' Bebras scores for the students with IEP/504 plans, $r(275) = 0.34, p < 0.0001$, and students without IEP/504 plans, $r(1004) = 0.41, p < 0.0001$. These correlations were not statistically different providing some evidence that the concurrent validity of the IACT does not differ by student IEP/504 status.

Table 12. Results for concurrent validity by student IEP/504 status

Correlations with IACT and Bebras measures	<i>Students with IEP/504s (N=275)</i>	<i>Students without IEP/504s (N=1004)</i>
Aggregated CT	0.34	0.41

Note: No differences between correlations were significant at an alpha level of 0.05.

5. Discussion

Validated measures of CT practices were needed to conduct research on game-based learning with the CT learning game, *Zoombinis*. The target audience included learners with IEPs who may have difficulty with textual assessment items and/or have no pre-existing knowledge of any type of coding language (including block-style introductory coding). Because of this we designed the IACT items based on upon similar models from psychological assessments that are typically used with a neurodiverse audience. In one case, for Pattern Recognition, we used an instrument drawn directly from clinical usage, the Raven's Progressive Matrices (Raven, 2000). For the other CT practices, we modified common interactive logic puzzles that used little to no text and required no previous coding experience. These items were designed for use in the *Zoombinis* study, and while they are not exactly aligned with the CT practices themselves, may provide a model for how more generalizable puzzles can be used to assess CT practices with young and neurodiverse learners and outside coding and computer science examples. The IACT items are intended to measure the four fundamental CT practices—Problem Decomposition, Pattern Recognition, Abstraction, and Algorithm Design—that were most evident in students' *Zoombinis* gameplay.

To explore the validity of these items, data were collected from two samples, along with external measures. The first sample included over 2500 elementary- and middle-school students who took part in a game-based learning study for the game *Zoombinis*. For this sample, we collected IACT data as well as teachers' CT ratings of their students on the same CT practices as IACT. We found with this sample that IACT items showed promise to measure CT, but we lacked a solid external measure for validation. Thus, we extended the study to include a second sample from a district-wide study where assessments were administered at the end of two different school years. For this sample, we collected IACT data as well as teachers' CT ratings of their students on the same CT practices as IACT, and we added Bebras items that aligned with the CT practices that were also collected from middle-school students.

The first hypothesis we studied was that the IACT demonstrated construct validity and thus could independently measure the four practices of CT. This hypothesis was confirmed. The items for each of the CT practices showed distinct results.

The second hypothesis we studied was that the IACT demonstrated a reliable measure over time. The test-retest reliability results between the pre- and post-tests for the individual CT practices were not strong enough to make a clear argument that learners perform consistently on these items for individual practices over time. Findings related to the aggregated measure of CT, however, indicated moderate test-retest reliability suggesting that this is a more consistent measure to use than items related to individual CT practices. This finding suggests that using an aggregated measure of CT can be appropriate for examining change in students' overall CT practices between two different points in time.

In confirmation of the third hypothesis, the aggregated CT assessment showed moderate evidence of concurrent validity. Our research correlated the IACT items to other external measures of these CT practices—a teacher CT rating sheet and Bebras items for the middle-school students in the RPP sample. Learners' overall performance aggregated across the four CT practices correlated with the teacher CT ratings of their students and the students' Bebras scores enough to make an argument for concurrent validity of the IACT items as an overall measure of CT. More refinement is needed for the IACT measures before they could serve as distinct assessments of individual CT practices, and it may be that these practices have too much overlap for distinction.

The final hypothesis, that the reliability and validity of IACT would not differ by student IEP/504 status, was confirmed across all three analyses. This supports our decision to design IACT items that were interactive (instead of multiple choice) and relied on limited text.

6. Conclusions

These findings suggest that IACT shows promise to contribute to the field of CT assessment but needs refinement to reach strong validity. In this current research, we have demonstrated moderate test-retest reliability and concurrent validity, and low to moderate construct validity for an aggregated measure of CT. IACT may be able to be further refined to distinguish and assess individual CT practices with future research.

As the field of CT education rapidly moves forward, it is important to establish a body of learning assessments that adequately measure students' practices associated with CT. In particular, it is important that these assessments are designed to capture the strengths and weaknesses demonstrated by a broad range of learners, including learners who may struggle with textual assessments and who have no pre-existing coding experience. This suggests the need for CT assessments that can measure practices without relying on text or coding. The IACT logic puzzles represent an important first step in this endeavor.

Not only are these items among the first with validation studies using a large number of learners, but they also have the unique strength of being designed with accessibility and learner variability in mind. The assessments extract information about students' CT practices in Problem Decomposition, Pattern Recognition, Abstraction, and Algorithm Design through students' activity in a set of logic puzzles as opposed to coding tasks or written questions. This work contributes not only to the field of measurement of CT, but also to the important task of finding inclusive ways to assess learning.

Limitations

There are several limitations to this study. Foremost is the lack of established external measures to which the validity of the IACT assessments can be compared. CT is an emergent field in K-12 education, and there are few assessment instruments for this age group and/or for learners with neurodiversity. IACT was designed for research in a game-based learning study that included neurodiverse students who may have not had previous experience with CT or coding as a target audience. The specific context of the research study also meant that the IACT items focus on four fundamental concepts of CT and do not attempt to define CT nor encompass all practices that could be included in CT. This assessment was designed to be administered in one class period, limiting the number of items for each CT construct. This likely played a role in the lower than typical correlations. Third, item type is confounded with CT construct (i.e., all items for a specific CT construct have the same unique item format), making a factor analysis of all CT items not meaningful (i.e., if items clustered by construct it could also be due to having a similar item type). Finally, multilevel analyses were not used in the reported study of the IACT items. While the data we used for these analyses has a nested structure, we did not have sufficient sample size at each level to adjust these correlations for this nestedness (e.g., students nested in courses nested in teachers). Future validation of IACT would need to account for variation among classes and teachers.

References

- Abdel-Khalek, A. M. (2005). Reliability and factorial validity of the standard progressive matrices among Kuwaiti children ages 8 to 15 years. *Perceptual and Motor Skills, 101*(2), 409–412.
- Allan, W., Coulter, B., Denner, J., Erickson, J., Lee, I., Malyn-Smith, J., & Martin, F. (2010). Computational Thinking for Youth. ITEST Small Working Group on Computational Thinking.
- American Association for the Advancement of Science. (1993). *Benchmarks for science literacy*. New York: Oxford University Press.
- Asbell-Clarke, J., Rowe, E., Almeda, V., Edwards, T., Bardar, E., Gasca, S., Baker, R.S., & Scruggs, R. (2020). The Development of Students' Computational Thinking Practices in Elementary- and Middle-School Classes using the Learning Game, *Zoombinis*. *Computers in Human Behavior*, <https://doi.org/10.1016/j.chb.2020.106587>
- Barendsen, E., Mannila, L., Demo, B., Grgurina, N., Izu, C., Mirolo, C., ... & Stupurienė, G. (2015, July). Concepts in K-9 computer science education. In *Proceedings of the 2015 ITiCSE on Working Group Reports* (pp. 85–116). ACM.
- Baron-Cohen, S., Ashwin, E., Ashwin, C., Tavassoli, T., & Chakrabarti, B. (2009). Talent in autism: hyper-systemizing, hyper-attention to detail and sensory hypersensitivity. *Philosophical Transactions of the Royal Society B: Biological Sciences, 364*(1522), 1377–1383.
- 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? *ACM Inroads, 2*(1), 48–54.

- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. Paper presented at the Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada.
- Computer Science Teachers Association. (2017). *CSTA K-12 Computer Science Standards*. Retrieved from <https://www.csteachers.org/page/standards>.
- Dagienė, V., & Futschek, G. (2008, July). Bebras international contest on informatics and computer literacy: Criteria for good tasks. In International conference on informatics in secondary schools-evolution and perspectives (pp. 19–30). Springer, Berlin, Heidelberg.
- Dagienė, V., Stupurienė, G., & Vinikienė, L. (2016, June). Promoting inclusive informatics education through the Bebras challenge to all K-12 students. In Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 (pp. 407–414). ACM.
- Dawson, M., Soulières, I., Ann Gernsbacher, M., & Mottron, L. (2007). The level and nature of autistic intelligence. *Psychological Science*, 18(8), 657–662.
- Dewey, J. (1938). *Logic, the theory of inquiry*. New York: Holt Pub.
- Duschl, R. A. (1990). *Restructuring science education: The importance of theories and their development*. Teachers College Press.
- González, M. R. (2015). Computational thinking test: Design guidelines and content validation. In Proceedings of EDULEARN15 conference (pp. 2436–2444).
- Grover, S., & Basu, S. (2017, March). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic. In Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education (pp. 267–272). ACM.
- Grover, S., Cooper, S., & Pea, R. (2014, June). Assessing computational learning in K-12. In Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 57–62). ACM.
- Grover, S., & Pea, R. (2013). Computational Thinking in K–12 A Review of the State of the Field. *Educational Researcher*, 42(1), 38–43.
- Haladyna, T. M., & Downing, S. M. (2004). Construct-irrelevant variance in high-stakes testing. *Educational Measurement: Issues and Practice*, 23(1), 17–27.
- Horowitz, S. H., Rawe, J., & Whittaker, M. C. (2017). *The State of Learning Disabilities: Understanding the 1 in 5*. New York: National Center for Learning Disabilities.
- Izu, C., Mirolo, C., Settle, A., Mannila, L., & Stupurienė, G. (2017). Exploring Bebras Tasks Content and Performance: A Multinational Study. *Informatics in Education*, 16(1), 39–59.
<https://files.eric.ed.gov/fulltext/EJ1140704.pdf>.
- Karalar, H., & Alpaslan, M. M. (2021). Assessment of Eighth Grade Students' Domain-General Computational Thinking Skills. *International Journal of Computer Science Education in Schools*, 5(1), 35 - 47.
<https://doi.org/10.21585/ijcses.v5i1.126>
- Koh, K. H., Basawapatna, A., Nickerson, H., & Repenning, A. (2014, July). Real time assessment of computational thinking. In *IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 49–52). IEEE.

- Kruit, P. M., Oostdam, R. J., van den Berg, E., & Schuitema, J. A. (2018). Assessing students' ability in performing scientific inquiry: instruments for measuring science skills in primary education. *Research in Science & Technological Education*, 1–27.
- Lundh, P., Grover, S., Jackiw, N., & Basu, S. (2018). Concepts Before Coding: Instructional Support for Introductory Programming Concepts in Middle School Computer Science. Annual Meeting of the American Education Research Association.
- Martinuzzi, A., & Krumay, B. (2013). The good, the bad, and the successful—how corporate social responsibility leads to competitive advantage and organizational transformation. *Journal of Change Management*, 13(4), 424–443.
- Mishra, P., Yadav, A., & Deep-Play Research Group. (2013). Rethinking technology & creativity in the 21st century. *TechTrends*, 57(3), 10–14.
- Moreno-León, J., & Robles, G. (2015, November). Dr. Scratch: a Web Tool to Automatically Evaluate Scratch Projects. In WiPSCE (pp. 132-133). https://www.researchgate.net/profile/Jesus_Moreno-Leon/publication/284181364_Dr_Scratch_a_Web_Tool_to_Automatically_Evaluate_Scratch_Projects/inks/564eccb508aefe619b0ff212.pdf.
- National Academy of Sciences on Computational Thinking (2010). Report of a Workshop on The Scope and Nature of Computational Thinking. National Academies Press.
- National Research Council (1996). National Science Education Standards. Washington, DC: The National Academies Press. p. 23. doi:10.17226/4962.
- O'Leary, U. M., Rusch, K. M., & Guastello, S. J. (1991). Estimating age-stratified WAIS-R IQS from scores on the Raven's standard progressive matrices. *Journal of Clinical Psychology*, 47(2), 277–284.
- Ota, G., Morimoto, Y., & Kato, H. (2016, September). Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts. In 2016 *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 238–239). IEEE.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.
- Papert, S. (1991). Situating constructionism. In I. Harel & S. Papert (Eds.), *Constructionism* (pp. 1-11). Norwood, NJ: Ablex.
- Raven, J. C. (1981). *Manual for Raven's progressive matrices and vocabulary scales. Research supplement No.1: The 1979 British standardisation of the standard progressive matrices and mill hill vocabulary scales, together with comparative data from earlier studies in the UK, US, Canada, Germany and Ireland*. San Antonio, TX: Harcourt Assessment.
- Raven, J.C., 2000. The Raven's progressive matrices: Change and stability over culture and time. *Cognitive Psychology*, 41(1), 1–48.
- Raven, J., Raven, J. C., & Court, J. H. (2000). *Manual for Raven's progressive matrices and vocabulary scales. Section 3: The standard progressive matrices*. Oxford, UK: Oxford Psychologists Press; San Antonio, TX: The Psychological Corporation.
- Ritchhart, R., Church, M., & Morrison, K. (2011). *Making thinking routines visible: How to promote engagement, understanding, and independence for all learners*. San Francisco, CA: Jossey-Bass.

- Román-González, M., Moreno-León, J., & Robles, G. (2017, July). Complementary tools for computational thinking assessment. In Proceedings of International Conference on Computational Thinking Education (CTE 2017), S. C Kong, J Sheldon, and K. Y Li (Eds.). The Education University of Hong Kong (pp. 154–159).
- 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. *Education and Information Technologies, 18*(2), 351–380.
- Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142–158.
- Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers and Education, 148*, [103798].
<https://doi.org/10.1016/j.compedu.2019.103798>
- U.S. Department of Education, Office of Special Education Programs, Individuals with Disabilities Education Act (IDEA) database, retrieved July 26, 2016, from <https://www2.ed.gov/programs/osepidea/618-data/state-level-data-files/index.html#bcc>. See Digest of Education Statistics 2016, table 204.30.
- von Wangenheim, C. G., Hauck, J. C., Demetrio, M. F., Pelle, R., da Cruz Alves, N., Barbosa, H., & Azevedo, L. F. (2018). CodeMaster--Automatic Assessment and Grading of App Inventor and Snap! Programs. *Informatics in Education, 17*(1), 117–150. <https://files.eric.ed.gov/fulltext/EJ1177148.pdf>.
- Wang, S. How Autism Can Help You Land a Job. *The Wall Street Journal*, March 27, 2014.
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127–147.
- Weintrop, D., Killen, H., Munzar, T., & Franke, B. (2019, February). Block-based Comprehension: Exploring and Explaining Student Outcomes from a Read-only Block-based Exam. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (pp. 1218–1224). ACM.
- Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012, February). 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). ACM.
<https://www.cs.auckland.ac.nz/courses/compsci747s2c/lectures/wernerFairyComputationalThinkingAssessment.pdf>.
- Wiebe, E., London, J., Aksit, O., Mott, B. W., Boyer, K. E., & Lester, J. C. (2019, February). Development of a Lean Computational Thinking Abilities Assessment for Middle Grades Students. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (pp. 456–461). ACM.
<https://dl.acm.org/citation.cfm?id=3287390>
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Appendices

Appendix A. Details of Data Cleaning

In the *Zoombinis* study, there were instances when teachers administered the pre-test later than expected. Specifically, students took the pre-test at a time when the post-test was supposed to be administered. In these cases, students' pre-test scores were treated as responses to the post-test measures. Students in these cases had missing pre-test scores. This rule was applied to 12 percent of our student sample ($n = 329$). The pattern of results did not vary when these students were removed, so they were retained for the analyses presented in this paper. It was possible for participants to complete some but not all of the items, so the total number of responses varied. Between 23 and 169 students had missing pre-test measures, depending on the CT practice. As many as 2,416 students (Study class students = 999, non-study class students = 1,417) had completed all pre-test measures belonging to at least one CT practice (Table 2) and were included in the reliability analyses. Of these 2,416 students, 1,523 students were from grades 3–5 (733 females, 790 males) and 893 students were from grades 6–8 (422 females, 468 males, 3 other).

For the RPP sample, between 22 and 255 students had missing pre-test measures. From a maximum number of 3,056 students with pre-test measures, 857 students were from grades 2–4 (429 females, 428 males) and 2,199 were from grades 5–8 (1,119 females, 1079 males, 1 other).

Table 1. Number of students who took the pre-assessment

	<i>Zoombinis</i>	RPP
Final Sample of Students	2,439	3,078
Students with Missing Pre-Assessment Scores*	23–169	22–255
Number of Students with Pre-Assessment Scores*	2,270–2,416	2,823–3,056

*Varies by CT Practice

In the *Zoombinis* sample, between 44 and 188 students had missing post-test measures, depending on the CT practice item set. A maximum of 1,773 students (study class students = 1016, non-study class students = 757) completed all post-assessment measures belonging to at least one CT practice (Table 2). Of these 1,773 students with complete post-assessment data in one CT practice, 1,174 students were from grades 3–5 (566 females, 608 males) and 599 students were from grades 6–8 (273 females, 323 males, 3 other). There were 1281 students who completed all pre-test and post-test measures across all CT practices (Study class students = 702, non-study class students = 579).

For the RPP sample, between 32 and 367 students had missing post-test measures. As many as 2,889 students completed all post-assessments belong to at least one CT practice (Table 3). From the 2,889 students with complete post-assessment data in one CT Practice, 1,147 students were from grades 2–4 (575 females, 572 males) and 1,742 students were from grades 5–8 (891 females, 850 males, 1 other).

Table 2. Number of students who took the post-assessment

	<i>Zoombinis</i>	RPP
Final Sample of Students	1,817	2,921
Students with Missing Post-Assessment Scores*	44–188	32–367
Number of Students with Post-Assessment Scores*	1,629–1,773	2,554–2,889

*Varies by CT Practice

Appendix B: Timed-out items for Problem Decomposition and Algorithm Design

Table 1: Number of students with pre-assessment and post-assessment items for Problem Decomposition (average efficiency)

	<i>Zoombinis</i> (pre-assessment)	<i>Zoombinis</i> (post-assessment)	RPP 2017-18	RPP 2018-19
Students with Complete Items for Problem Decomposition (average efficiency)	2,416	1,773	3,056	2,889
Students Who Timed Out of Assessment Items for Problem Decomposition (average efficiency)	106	51	180	142
Students Who Timed Out of All Assessment Items for Problem Decomposition (average efficiency)	23	44	22	32

Appendix 2. Number of students with pre-assessment items and post-assessments items for Algorithm Design (mean number of optimal moves)

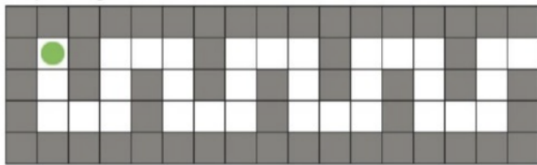
	<i>Zoombinis</i> (pre-assessment)	<i>Zoombinis</i> (post-assessment)	RPP 2017-18	RPP 2018-19
Students with Complete Assessment Items for Algorithm (average efficiency)	2,270	1,691	2,823	2,607
Students Who Timed Out of Pre-Assessment Items for Algorithm Design (average efficiency)	793	458	1,228	1,070
Students Who Timed Out of Post-Assessment Items for Algorithm Design (average efficiency)	169	41	255	314

Appendix C. Bebras Items

We selected 5 Bebras items that corresponded to our 4 types of logic puzzles. Items 3 and 4 are most similar to our Problem Decomposition items. Items 1, 2, and 5, and 1 were most similar to our Algorithm Design, Pattern Recognition, and Abstraction items, respectively.

Bebras Item 1

Help the green robot to exit the maze.




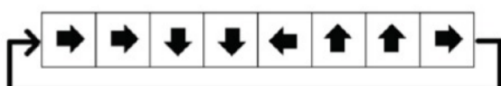
The arrows below represent the instructions that the green robot can follow.





Choose the correct set of instructions that will take the green robot to the exit. The robot will repeat these instructions 4 times.

Select the correct answer *

A.  4x

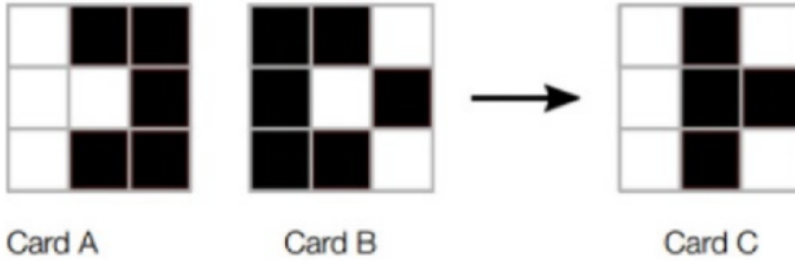
B.  4x

C.  4x

D.  4x

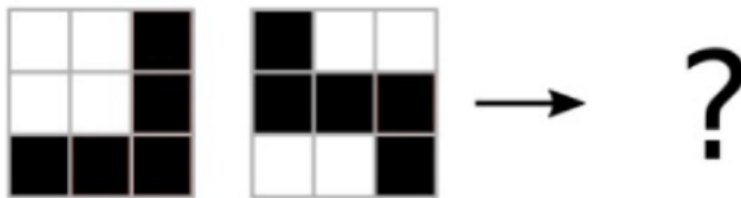
Bebras Item 2

Combining Card A and Card B, you get Card C:



Question:

How many black cells will Card F have after combining Card D and Card E?



• **Select the correct answer ***

- 3
- 4
- 5
- 6

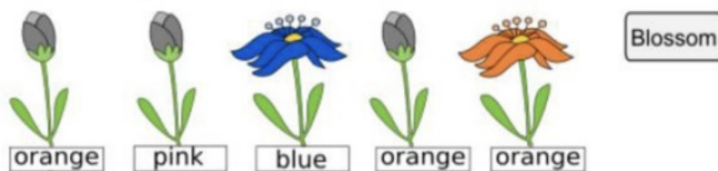
Bebras Item 3

Jane is playing a computer game.

First, the computer secretly chooses colors for five buds. The available colors for each flower are blue, orange, and pink. Jane has to guess which flower has which color. She makes her first five guesses and presses the Blossom button.

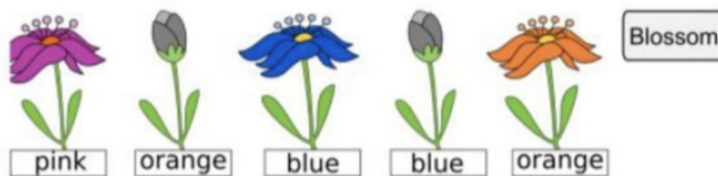
The buds, whose colors she guessed correctly, break into flowers. The others remain as buds.

Jane's first go:



Jane then has another go at guessing and presses the Blossom button again.

Jane's second go:



Question:

What colors did the computer choose for the flowers?

• **Select the correct answer ***

- | |
|---|
| <input type="checkbox"/> blue pink blue orange orange |
| <input type="checkbox"/> pink blue blue blue orange |
| <input type="checkbox"/> pink blue blue pink orange |
| <input type="checkbox"/> pink pink blue pink orange |

Bebras Item 4:

Betaro Beaver has discovered five new magic potions:

- one makes ears longer
- another makes teeth longer
- another makes whiskers curly
- another turns the nose white
- the last one turns eyes white.

Betaro put each magic potion into a separate beaker. He put pure water into another beaker, so there are six beakers in total. The beakers are labeled A to F. The problem is, he forgot to record which beaker contains which magic potion!

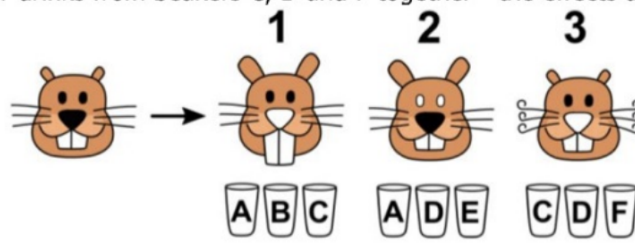


To find out which potion is in each beaker, Betaro set up the following experiments:

Expt 1: A beaver drinks from beakers A, B and C together - the effects are shown in Figure 1.

Expt 2: A beaver drinks from beakers A, D and E together - the effects are shown in Figure 2 .

Expt 3: A beaver drinks from beakers C, D and F together - the effects are shown in Figure 3.



Question:

Which beaker contains pure water?

• **Select the correct answer ***

<input type="checkbox"/>	A
<input type="checkbox"/>	B
<input type="checkbox"/>	C
<input type="checkbox"/>	D
<input type="checkbox"/>	E
<input type="checkbox"/>	F

Bebras Item 5

Agents Boris and Bertha communicate using secret messages.
Boris wants to send Bertha the secret message:
MEETBILLYBEAVERAT6

He writes each character in a 4 column grid from left to right and row by row starting from the top. He puts an X in any unused spaces. The result is shown below.



Then he creates the secret message by reading the characters from top to bottom and column by column starting from the left:
MBYVTEIBE6ELERXTLAAX

Bertha then uses the same method to reply to Boris. The secret message she sends him is:
OIERKLTEILH!WBEX

Question:
What message does Bertha send back?

• **Select the correct answer ***

<input type="checkbox"/> OKWHERE TOMEET!
<input type="checkbox"/> OKIWILLBETHERE!
<input type="checkbox"/> WILLYOUBETHERETOO?
<input type="checkbox"/> OKIWILLMEETHIM!

Appendix D. Construct Validity

For the Problem Decomposition and Algorithm Design items, we tried several means of scoring those items.

These included the *total number of moves* the player used to solve the problem, *percentage of items solved correctly*, the *efficiency* of the number of moves players used relative to the maximum number of moves needed

to find a solution, and the *percentage of items solved optimally*. Because students could make more than one attempt to solve each Algorithm Design item, the *total number of runs* they made was also examined.

Percent Correct: Whether or not each item was answered correctly, regardless of the number of moves, was recorded. It was possible for students to time out of each item without answering correctly. The percentage of items answered correctly was calculated.

Efficiency: For Problem Decomposition items, a maximum of 3 moves was needed to solve elementary problems while 4 moves were needed to solve the middle-school problems. The efficiency with which each item was solved was calculated by dividing this maximum number by the actual number of moves taken. In cases where players were able to solve the problem with fewer than the maximum number of moves, their efficiency was capped at 100 percent. With Algorithm Design items, the minimum number of moves needed to find a solution depended on the number of submissions players made. This minimum number of moves was recorded for each item.

Optimal Solutions: Optimal solutions for Problem Decomposition items were those solved with 100 percent efficiency. For Algorithm Design items, a solution was considered optimal if it was solved with 100 percent efficiency with one submission.

Tables 1 and 2 report the correlations between all of the scoring approaches taken for the Problem Decomposition and Algorithm Design items with each other and with the Pattern Recognition and Abstraction item percent correct scores.

Table 1. Pearson intercorrelations of pre-assessment measures for the *Zoombinis* and RPP samples

Correlations between CT Practices	CT Practice	<i>Zoombinis</i> sample (N=2206-2416)	RPP sample (N=2772-3056)	Average across samples
Problem Decomposition (# Moves)	Problem Decomposition (Percent Optimal)	-0.80	-0.77	-0.79
	Problem Decomposition (Percent Correct)	-0.18	-0.25	-0.22
	Problem Decomposition (Avg. Efficiency)	-0.84	-0.79	-0.81
	Pattern Recognition (Correct)	-0.15	-0.12	-0.13

	Abstraction (Percent Correct Spaces)	-0.21	-0.11	-0.16
	Algorithm Design (# Moves)	-0.06	-0.04	-0.05
	Algorithm Design (Percent Correct)	-0.23	-0.07	-0.15
	Algorithm Design (# Runs)	0.21	0.07	0.14
	Algorithm Design (Avg. Efficiency)	-0.23	-0.08	-0.15
	Algorithm Design (Percent Optimal)	-0.19	-0.04	-0.12
Problem Decomposition (Percent Optimal)	Problem Decomposition (Percent Correct)	0.24	0.33	0.29
	Problem Decomposition (Avg. Efficiency)	0.89	0.87	0.88
	Pattern Recognition (Correct)	0.18	0.13	0.15
	Abstraction (Percent Correct Spaces)	0.22	0.16	0.19
	Algorithm Design (# Moves)	0.08	0.06	0.07
	Algorithm Design (Percent Correct)	0.22	0.10	0.16
	Algorithm Design (# Runs)	-0.21	-0.08	-0.14
	Algorithm Design (Avg. Efficiency)	0.23	0.11	0.17
	Algorithm Design (Percent Optimal)	0.20	0.07	0.14

Problem Decomposition (Percent Correct)	Problem Decomposition (Avg. Efficiency)	0.54	0.65	0.60
	Pattern Recognition (Correct)	0.07	0.04	0.06
	Abstraction (Percent Correct Spaces)	0.10	0.09	0.09
	Algorithm Design (# Moves)	0.11	0.09	0.10
	Algorithm Design (Percent Correct)	0.14	0.13	0.13
	Algorithm Design (# Runs)	-0.07	0.01	-0.03
	Algorithm Design (Avg. Efficiency)	0.14	0.13	0.13
	Algorithm Design (Percent Optimal)	0.06	-0.01	0.03
Problem Decomposition (Avg. Efficiency)	Pattern Recognition (Correct)	0.18	0.12	0.15
	Abstraction (Percent Correct Spaces)	0.23	0.16	0.20
	Algorithm Design (# Moves)	0.10	0.08	0.09
	Algorithm Design (Percent Correct)	0.26	0.13	0.20
	Algorithm Design (# Runs)	-0.22	-0.07	-0.15
	Algorithm Design (Avg. Efficiency)	0.27	0.14	0.21
	Algorithm Design (Percent Optimal)	0.21	0.05	0.13

Pattern Recognition (Correct)	Abstraction (Percent Correct Spaces)	0.32	0.30	0.31
	Algorithm Design (# Moves)	0.11	0.15	0.13
	Algorithm Design (Percent Correct)	0.28	0.24	0.26
	Algorithm Design (# Runs)	-0.26	-0.21	-0.23
	Algorithm Design (Avg. Efficiency)	0.24	0.21	0.23
	Algorithm Design (Percent Optimal)	0.25	0.10	0.18
Abstraction (Percent Correct Spaces)	Algorithm Design (# Moves)	0.09	0.12	0.11
	Algorithm Design (Percent Correct)	0.28	0.25	0.27
	Algorithm Design (# Runs)	-0.26	-0.24	-0.25
	Algorithm Design (Avg. Efficiency)	0.26	0.26	0.26
	Algorithm Design (Percent Optimal)	0.24	0.16	0.20
Algorithm Design (# Moves)	Algorithm Design (Percent Correct)	0.34	0.45	0.40
	Algorithm Design (# Runs)	-0.12	-0.11	-0.11
	Algorithm Design (Avg. Efficiency)	0.11	0.35	0.23
	Algorithm Design (Percent Optimal)	-0.08	0.07	-0.01
Algorithm Design (Percent Correct)	Algorithm Design (# Runs)	-0.41	-0.29	-0.35

	Algorithm Design (Avg. Efficiency)	0.96	0.98	0.97
	Algorithm Design (Percent Optimal)	0.43	0.32	0.37
Algorithm Design (Runs)	Algorithm Design (Avg. Efficiency)	-0.40	-0.30	-0.35
	Algorithm Design (Percent Optimal)	-0.64	-0.51	-0.57
Algorithm Design (Avg. Efficiency)	Algorithm Design (Percent Optimal)	0.50	0.36	0.43

Note: Significant at an alpha level of 0.05 except if in italics.

Table 2. Pearson intercorrelations of post-assessment measures for *Zoombinis* and RPP samples

Correlations between CT Practices	CT Practice	<i>Zoombinis</i> sample (N=1600-1774)	RPP sample (N=2315-2889)	Average across samples
Problem Decomposition (# Moves)	Problem Decomposition (Percent Optimal)	-0.76	-0.65	-0.71
	Problem Decomposition (Percent Correct)	-0.11	-0.12	-0.12
	Problem Decomposition (Avg. Efficiency)	-0.83	-0.67	-0.75
	Pattern Recognition (Correct)	-0.14	-0.11	-0.13
	Abstraction (Percent Correct Spaces)	-0.19	-0.13	-0.16
	Algorithm Design (# Moves)	-0.02	-0.05	-0.04

	Algorithm Design (Percent Correct)	-0.22	-0.09	-0.16
	Algorithm Design (# Runs)	0.23	0.13	0.18
	Algorithm Design (Avg. Efficiency)	-0.20	-0.09	-0.15
	Algorithm Design (Percent Optimal)	-0.17	-0.06	-0.12
Problem Decomposition (Percent Optimal)	Problem Decomposition (Percent Correct)	0.23	0.25	0.24
	Problem Decomposition (Avg. Efficiency)	0.90	0.72	0.81
	Pattern Recognition (Correct)	0.20	0.13	0.17
	Abstraction (Percent Correct Spaces)	0.24	0.12	0.18
	Algorithm Design (# Moves)	-0.02	0.08	0.03
	Algorithm Design (Percent Correct)	0.21	0.11	0.16
	Algorithm Design (# Runs)	-0.25	-0.08	-0.17
	Algorithm Design (Avg. Efficiency)	0.21	0.11	0.16
	Algorithm Design (Percent Optimal)	0.21	0.09	0.15
Problem Decomposition (Percent Correct)	Problem Decomposition (Avg. Efficiency)	0.48	0.70	0.59

	Pattern Recognition (Correct)	0.08	0.05	0.07
	Abstraction (Percent Correct Spaces)	0.09	0.12	0.11
	Algorithm Design (# Moves)	0.00	0.05	0.03
	Algorithm Design (Percent Correct)	0.09	0.07	0.08
	Algorithm Design (# Runs)	-0.06	0.03	-0.02
	Algorithm Design (Avg. Efficiency)	0.09	0.06	0.08
	Algorithm Design (Percent Optimal)	0.06	0.03	0.05
Problem Decomposition (Avg. Efficiency)	Pattern Recognition (Correct)	0.19	0.12	0.16
	Abstraction (Percent Correct Spaces)	0.24	0.20	0.22
	Algorithm Design (# Moves)	-0.01	0.10	0.05
	Algorithm Design (Percent Correct)	0.23	0.14	0.19
	Algorithm Design (# Runs)	-0.25	-0.14	-0.20
	Algorithm Design (Avg. Efficiency)	0.22	0.14	0.18
	Algorithm Design (Percent Optimal)	0.21	0.09	0.15
Pattern Recognition (Correct)	Abstraction (Percent Correct Spaces)	0.35	0.31	0.33
	Algorithm Design (# Moves)	0.08	0.15	0.12

	Algorithm Design (Percent Correct)	0.36	0.30	0.33
	Algorithm Design (# Runs)	0.27	0.26	0.27
	Algorithm Design (Avg. Efficiency)	0.24	0.24	0.24
	Algorithm Design (Percent Optimal)	0.20	0.17	0.19
Abstraction (Percent Correct Spaces)	Algorithm Design (# Moves)	0.08	0.13	0.11
	Algorithm Design (Percent Correct)	0.28	0.23	0.26
	Algorithm Design (# Runs)	-0.23	-0.20	-0.22
	Algorithm Design (Avg. Efficiency)	0.24	0.23	0.24
	Algorithm Design (Percent Optimal)	0.22	0.21	0.22
Algorithm Design (# Moves)	Algorithm Design (Percent Correct)	0.33	0.47	0.40
	Algorithm Design (# Runs)	-0.09	-0.09	-0.09
	Algorithm Design (Avg. Efficiency)	0.09	0.37	0.23
	Algorithm Design (Percent Optimal)	-0.12	-0.08	-0.10
Algorithm Design (Percent Correct)	Algorithm Design (# Runs)	-0.35	-0.32	-0.34
	Algorithm Design (Avg. Efficiency)	0.95	0.98	0.97
	Algorithm Design (Percent Optimal)	0.38	0.42	0.40

Algorithm Design (Runs)	Algorithm Design (Avg. Efficiency)	-0.35	-0.33	-0.34
	Algorithm Design (Percent Optimal)	-0.66	-0.57	-0.62
Algorithm Design (Avg. Efficiency)	Algorithm Design (Percent Optimal)	0.47	0.47	0.47

Note: Significant at an alpha level of 0.05 except if in italics.

Appendix E. Concurrent Validity

Table 1. Correlations between post-test measures and teacher ratings of their students' CT skills

CT Practice	Problem Decomposition	Pattern Recognition	Abstraction	Algorithm Design
Problem Decomposition (Moves)	-0.13	-0.11	-0.07	-0.10
Problem Decomposition (Percent Optimal)	0.16	0.15	0.10	0.15
Problem Decomposition (Percent Correct)	0.07	0.08	<i>0.04</i>	<i>0.04</i>
Problem Decomposition (Avg. Efficiency)	0.17	0.15	0.10	0.14
Pattern Recognition (Percent Correct)	0.28	0.23	0.19	0.23
Abstraction (Percent Correct Spaces)	0.23	0.15	0.13	0.16
Algorithm Design (#Moves)	<i>-0.03</i>	-0.08	<i>-0.05</i>	<i>-0.04</i>
Algorithm Design (Percent Correct)	0.15	0.09	0.08	0.12
Algorithm Design (# Runs)	-0.16	-0.14	-0.09	-0.13
Algorithm Design (Avg. Efficiency)	0.18	0.14	0.13	0.17
Algorithm Design (% Optimal)	0.23	0.21	0.18	0.22

Note: Significant at an alpha level of 0.05 except if in italics. $N=892-944$ depending on the measures.

As shown in Table 2, IACT logic puzzle items for Algorithm Design (Moves) were weakly correlated with Bebras items in an unexpected positive direction. As Algorithm Design (Moves) is more indicative of persistence than correctness, it is possible that students who persisted and had a great number of moves in IACT also had lower scores in each of the five Bebras items.

Table 2. Correlations between post-test measures and students' scores on Bebras items

CT Practice	Problem Decomposition (Item 3)	Problem Decomposition (Item 4)	Pattern Recognition (Item 2)	Abstraction (Item 5)	Algorithm Design (Item 1)
Problem Decomposition (Moves)	-0.08	-0.09	<i>-0.04</i>	-0.11	-0.10
Problem Decomposition (Percent Optimal)	0.07	0.13	<i>0.04</i>	0.10	0.07
Problem Decomposition (Percent Correct)	<i>0.00</i>	<i>0.05</i>	<i>-0.01</i>	<i>0.01</i>	0.07
Problem Decomposition (Avg. Efficiency)	0.05	0.13	<i>0.04</i>	0.10	0.12
Pattern Recognition (Percent Correct)	0.18	0.15	0.10	0.13	0.17
Abstraction (Percent Correct Spaces)	0.23	0.22	0.12	0.22	0.28
Algorithm Design (#Moves)	<i>0.05</i>	0.08	0.06	0.09	0.07
Algorithm Design (Percent Correct)	0.11	0.13	0.09	0.09	0.24
Algorithm Design (# Runs)	-0.10	-0.09	-0.07	-0.13	-0.14
Algorithm Design (Avg. Efficiency)	0.11	0.12	0.08	0.09	0.24
Algorithm Design (% Optimal)	0.11	0.08	0.07	0.11	0.13

Note: Significant at an alpha level of 0.05 except if in italics. $N=1,243-1,399$.

Appendix F. Concurrent validity of external CT measures

Table 1. Correlations between teacher ratings of their students' CT skills

	Pattern Recognition	Abstraction	Algorithm Design
	0.78	0.75	0.75

Problem Decomposition			
Pattern Recognition		0.77	0.70
Abstraction			0.70

Note: Significant at an alpha level of 0.0001; $N=1,091$.

Table 2. Correlations between Bebras items

CT Practice	Problem Decomposition (Item 4)	Pattern Recognition (Item 2)	Abstraction (Item 5)	Algorithm Design (Item 1)
Problem Decomposition (Item 3)	0.15	0.11	0.12	0.13
Problem Decomposition (Item 4)		0.12	0.13	0.09
Pattern Recognition (Item 2)			0.06	0.07
Abstraction (Item 5)				0.09

Note: Significant at an alpha level of 0.05; $N=1,355-1,387$.

Good examples help; bad tools hurt: Lessons for teaching computer security skills to undergraduates

Jonathan Sharman, Claudia Ziegler Acemyan, Philip Kortum, Dan Wallach

Rice University, USA

DOI:

Abstract

Software security is inevitably dependent on developers' ability to design and implement software without security bugs. Perhaps unsurprisingly, developers often fail to do this. Our goal is to understand this from a usability perspective, identifying how we might best train developers and equip them with the right software tools. To this end, we conducted two comparatively large-scale usability studies with undergraduate CS students to assess factors that affect success rates in securing web applications against cross-site request forgery (CSRF) attacks. First, we examined the impact of providing students with example code and/or a testing tool. Next, we examined the impact of working in pairs. We found that access to relevant secure code samples gave significant benefit to security outcomes. However, access to the tool alone had no significant effect on security outcomes, and surprisingly, the same held true for the tool and example code combined. These results confirm the importance of quality example code and demonstrate the potential danger of using security tools in the classroom that have not been validated for usability. No individual differences predicted one's ability to complete the task. We also found that working in pairs had a significant positive effect on security outcomes. These results provide useful directions for teaching computer security programming skills to undergraduate students.

Keywords: Security, coding, teaching, usability, tools

1. Introduction

Despite a growing emphasis among security experts on secure coding practices, software developers continue to regularly misuse or misunderstand secure coding tools. Understanding how to best train students in good security coding practices is critical to designing safer software. Recent efforts within the area of usable security research have attempted to enumerate causes for developer error leading to security vulnerabilities in software. For example, access to good documentation and reliable example code have significant impacts on solving security tasks (Acar et al., 2017; Fischer et al., 2017; Mindermann and Wagner, 2018; Mindermann and Wager, 2020), as does priming (Naiakshina et al., 2018). Usability issues can also impact the appropriate use of other security-related systems, including Android development (Acar et al., 2016), cryptographic APIs (Acar et al., 2017; Acar

et al., 2017; Gorski et al., 2018; Naiakshina et al., 2019; Oliveira et al., 2018; Zeier et al., 2019), type systems (Weber et al., 2017), HTTPS deployment (Krombholz et al., 2017; Bernhard et al., 2019), OpenSSL (Ukrop and Matyas, 2018), and string and I/O APIs (Oliveira et al., 2018).

Our work builds on previous studies by trying to understand how to better instruct undergraduate computer science students in the art of security programming, examining the impact of using code samples, software tools, and group programming in the classroom. We wanted to determine if there were benefits in providing students with software code examples and software tools that would aid them in testing for code security. We also wanted to determine if working in teams as students was beneficial in their learning efforts. We chose cross-site request forgery (CSRF) as a security problem for our study because it's still relevant to current practice and because of its relative simplicity for use as a teaching tool. We performed two between-subjects usability studies, one year apart, with undergraduate computer science students. In study 1 we examined the impacts of example code and a CSRF detection tool on a student's ability to repair CSRF vulnerabilities in a test server. In study 2, we examined the effects of the students working alone vs. working in pairs.

2. Study 1: Impact of example code and software tools

2.1 Methodology

We drew our students from the sophomore-level COMP 215 class at Rice University, a highly-selective four-year residential college. Rice University had a total sophomore population of approximately 1,000 students, and Computer Science is the most popular major on campus, with approximately 19% of 2018 sophomores enrolling in COMP 215. COMP 215 students present a relatively uniform pool of students. Most COMP 215 students have never taken a college-level course in Java before, having come to Rice University immediately after high school. (Only eight students among the students in this study were transfer students.) All COMP 215 students take the same freshman class sequence, including an introductory course in Python, and a theory course. To help prepare students for the security task, the COMP 215 lectures during the week of the experiment considered "Web 2.0" designs (e.g., Java microservices with JavaScript clients). Students were strongly encouraged to attend a security-specific lecture as well.

2.1.1 Design

This study used a between-subjects design with two variables: (1) availability of a CSRF testing tool and (2) availability of an example web application (code) with CSRF mitigation, resulting in four experimental conditions. The "no tool, no example" group, which served as the control group, had access to neither CSRF Testing Tools nor to the example code. The "example-only" group had access to the example code but not to the tool. The "tool-only" group had access to the tool but not to the example code. Finally, the "tool and example" group had access to both the tool and the example code. Students were randomly assigned into one of the four conditions. Students in all groups, including those not provided with example code, were allowed to search the Internet for examples and instructions but not to ask anyone for help.

We selected CSRF prevention as the task for this study because it is a realistic security problem but also relatively simple, allowing us to teach the relevant concepts to our students over a few lectures and construct a short, self-contained study based on the problem. In contrast, many other security vulnerabilities, such as buffer overflows, can be very subtle to understand or even recognize when looking at the code.

Condition Assignment. Consistent with the other programming assignments in COMP 215, students attempted this task individually. (We examine pair programming in Study 2, below.) We controlled for gender and midterm grades when assigning students to conditions, to reduce possible confounding factors and to allow us to check for any effects these variables had on learning outcomes. We partitioned the remaining students by gender and sorted both lists by midterm grade. We then pulled students in blocks of four and distributed them at random among the four groups. Students who did not identify as male or female were assigned to groups at random. Despite a small number of drop-outs after experimental group assignment, this strategy yielded a consistent demographic distribution, with 44 participants assigned to the control group, 46 to the “example-only” group, 45 to the “tool-only” group, and 45 to the “tool and example” group. Students were instructed not to discuss or share any aspect of the assignment with each other, including their condition assignment, and since the project was graded only on participation, they had no academic incentive to disclose their condition assignment to other students. Average age of the students was 19.2 years (SD 0.8). Each of the groups was composed of approximately 28% females. Java experience was consistent across groups, with an average of 2.1 on a 5-point scale, where 5 is expert (SD 1.0).

The dependent variables in this study were effectiveness (how well the students completed the task) and efficiency (how long it took them to complete the task) in implementing CSRF prevention in a test server. For groups with access to the CSRF Testing Tool, we also measured satisfaction with the provided tool, i.e. how well the tool met the student’s expectations. Effectiveness, efficiency, and satisfaction are three standard measures used to assess system usability per ISO 9241-11.

2.1.2 Measures

Effectiveness. Effectiveness is the degree of success in achieving a goal (ISO 9241-11). Consistent with previous assignments in the course, we assigned each student’s work a security score from 0 to 10 using a subtractive grading rubric (Table 1). Deductions were capped at 10 points; i.e., a solution with 10 or more deductions received a security score of 0. Most mistakes resulted in a one-point deduction, with the exception of failing to employ any kind of server-provided key, which resulted in a four-point deduction.

One of the most common anti-CSRF techniques consists of sending a randomized token from the server to the client, returning that token from the client to the server with each request, and verifying on the server that the tokens match. Critically, the client must not return the token via a cookie since cookies associated with a domain are sent automatically with each request to that domain. While students were permitted to use whatever method they wanted to complete the task, this is the technique students were taught during lecture and the approach they generally took.

Table 1. Security score rubric

Error	Deduction
Server vulnerable to GET-based CSRF	1
No server-generated key	4
Key entropy < 32 bits	1
Key entropy < 64 bits	1

No handshake mechanism to prevent back doors	1
Vulnerable to timing attacks	1
Server does not attempt to send key	1
Client does not attempt to retrieve key	1
Server-to-client methods do not match	1
Key not delivered server-to-client	1
Client does not send key for validation	1
Server does not request key for validation	1
Client-to-server methods do not match	1
Key not sent client-to-server	1
Client sends key insecurely (e.g. via a cookie)	1
Server does not validate received key	1

We initially considered a scoring system based only on the presence of actual vulnerabilities; however, we decided that such a system did not capture how well each student understood the problem or how secure their solution was. For example, many students submitted solutions that were correct except that they did not use a one-time key to prevent a fraudulent web form from retrieving the CSRF-prevention key, leaving the server vulnerable. These students might receive a zero score from a classical security analysis, despite demonstrating significant progress towards a secure solution. Therefore, we instead chose a scoring system based on the accumulation of individual errors. While students were free to implement any CSRF mitigation mechanism they chose, the vast majority used techniques similar to those described in both the lecture materials and the example code.

Additionally, we assigned each student a passing or failing functionality score based on whether they broke the original functionality of the system. For example, some students made it impossible to perform any transactions at all. For the sake of simplicity, we graded solutions with minor changes in functionality (such as reduced responsiveness) as functionally correct.

Timing. We set the maximum duration for the study to 180 minutes (measured to the nearest minute), opting for this duration both to respect the students' time and to enable most students to complete the study in a single sitting. The time on task does not include reading the project specification prior to beginning the task or the post-study survey. We obtained timing data in two ways, in an attempt to improve the reliability of the timing data: self-reported time on task and git push/commit times. Even so, it was difficult to obtain high-quality timing data under our experimental setup. In cases where the two measurements did not align or where the available timing data was clearly wrong (for instance, a full day spent on the task, based on push times), we had to discard the data for the purpose of timing analysis.

Surveys. We used SurveyMonkey for both the demographic and post-study surveys. For those in the experimental groups that had access to CSRF Testing Tools, the post-study survey includes a System Usability Scale (SUS) (Brooke, 1996) evaluation in the final survey to measure satisfaction with the tool. The SUS is one of the most commonly used, psychometrically validated (Bangor et al., 2008) tools that measure the usability of products,

services, and systems. SUS scores range from zero (unusable) to 100 (highly usable).

2.1.3 Materials

Spark Java Server. Spark Java¹ is a simple Java web framework. We wrote a Spark Java server and client for a toy application called “Fruit Market”, which simulates buying and selling of fruit with fluctuating prices. Students run both the server and client locally, changing the server state by clicking the buy and sell buttons. We intentionally left the server open to CSRF attacks by implementing the buying and selling functionality using simple GET requests. For instance, a purchase of a certain type of fruit can be made by issuing a GET request to `/buy/?index=0`, and a sale of the same fruit can be made through `/sell/?index=0`. Since there is no CSRF mitigation mechanism, the application is vulnerable to all the standard CSRF attacks, such as embedding these URLs in an HTML image element. Note that Spark Java does not include any CSRF prevention mechanism.

Handouts / GitHub Classroom. GitHub Classroom allows instructors to create assignments from template git repositories, which students clone when they click on a given link. We created four project PDFs and four Classroom clone links, corresponding to our four experimental conditions, written in the same style as our regular weekly project assignments.

Lectures. We presented a three-lecture sequence considering how web browsers and servers operate, along with the security issues they face. We provide PDFs of all lecture slides on our course web page.

Example Code. Half of the students had access to an example web application, already secure against CSRF attacks. The example server, also written in Spark Java, implemented the back-end for a JavaScript read-eval-print-loop (REPL). The JS REPL server avoids CSRF by launching the user’s browser with a one-time key in the URL, which the client then returns to the server in order to retrieve a CSRF-prevention key, generated randomly at server launch. This key then serves to validate subsequent calls to the server.

CSRF Testing Tools. Half of the students had access to CSRF Testing Tools², a free, semi-automated CSRF exploit generation tool, along with the instructions provided by the tool’s author. The tool consists of two parts: a “FormGrabber” bookmarklet that can be used to copy HTML form content from a webpage into the user’s clipboard and a “FormBuilder” page that creates a spoofed form from the copied form content. While the README describes how to use the tool to “create [HTML] forms that mimic the forms on the site that you’re testing”, it does not explain how to interpret the results and whether they indicate a CSRF vulnerability. Indeed the README states “I am assuming that you have a good understanding of what a CSRF attack is and can figure out how this tool mimics one. Explaining the anatomy of a CSRF attack is not something I’m going to do in this documentation.” In short, the ability to change server-side application state via interaction with the spoofed form indicates a CSRF vulnerability.

Students had no prior experience with this tool in the course but were instructed to read the included README describing the steps required to set up and utilize the tool to check for CSRF vulnerabilities. As part of its setup process, CSRF Testing Tools requires (1) hosting a JavaScript file on a server separate from the server under test and (2) modifying a bookmarklet file to point to the hosted file. We gave the students a copy of the latest version of the tool as found on GitHub, with one modification: we hosted the file for them and gave them a

¹ <https://sparkjava.com/>

² <https://github.com/akrikos/CSRF-Testing-Tools>

pre-modified version of the bookmarklet. While performing this modification ahead of time changes the experience of using the tool, we determined that requiring the students to complete this step themselves would be unreasonably difficult and time-consuming within the limits of the study. As our results show, the tool's measured effectiveness and satisfaction were low, even with this simplification to the process.

We considered several other free CSRF prevention tools—including Burp Suite³, CSRFTester⁴, Pinata⁵, CSRF PoC Generator⁶, and OWASP Zed Attack Proxy (ZAP)⁷—but in our judgment, CSRF Testing Tools was the most useful among these options.

An anonymized repository with experiment materials can be found at <https://github.com/bad-tools-hurt/csrf>. This repository contains the code, handouts, and lectures used in both studies, with all references to the authors and their institution(s) removed. We have removed code that is both irrelevant to the task and unique to the course. In particular, for this repository, we replaced a course-specific JSON library with a third-party library.

2.1.4 Procedures

As with any human subjects experiment, we obtained IRB approval before beginning the study. We recruited a total of 194 students from the fall 2018 COMP 215 class. Students received partial course credit for participating (regardless of their success) and had the ability to opt out of the study at any point prior to or during the study. None of the students opted out; however, a total of 14 students either dropped the course prior to the completion of the study or otherwise did not complete the study, leaving a total of 180 students in the data set.

Once the assignment was given, the students could choose to attempt the three-hour task any time within a five-day period. We encouraged the students to attempt the task in one sitting, but they were allowed to take breaks as long as they noted the stop and start times via git commit messages. The students had to complete the following steps: (1) read the project specification PDF; (2) click the GitHub Classroom assignment link (embedded in the PDF), to set up their repository; (3) add their name and student ID to a README file prior to beginning the task and then commit and push to GitHub (the timestamp of this initial push allowed us to measure the start of the time on task); (4) secure the test server against CSRF attacks, by whatever method they choose, within 180 minutes of their initial push time; (5) after completing the task, running into the time limit, or giving up on the task, commit and push their final code; and (6) fill out the post-study survey.

We chose to have the students work on their own time in a setting of their choosing, rather than in a lab, for better scalability and to allow the students to complete the task under their usual work conditions, making comparisons with past course performance more meaningful. The first author manually graded each submission according to our security and functionality rubrics. We used the CSRF Testing Tools to speed up the grading process; however, it was only useful as a first step towards measuring security. For instance, many students came very close to a correct solution but did not use a handshake mechanism, such as a one-time key, to prevent a fraudulent web form

³ <https://portswigger.net/burp>

⁴ <https://github.com/tomasperezv/web-security-tools/tree/master/CSRFTester>

⁵ <https://github.com/ahsansmir/pinata-csrf-tool>

⁶ <https://security.love/CSRF-PoC-Genorator/>

⁷ https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

from reading the anti-CSRF key from the HTML. Under our rubric, such a solution is graded as 9/10, but its security-relevant behavior is identical to a solution with no CSRF mitigation at all. Conversely, some solutions were invulnerable to the particular attack generated by CSRF Testing Tools but used insufficient key entropy and thus still lost points. Therefore, we still had to manually inspect each submission in order to understand whether a solution was fully secure and, if not, how many points should be deducted.

2.2 Results and Discussion

We performed an ANOVA for security score, functionality score, and time on task, with experimental condition assignment (group) and gender as factors, using Tukey’s HSD to correct for multiple comparisons.

Security Scores. The mean security score was 1.43 for the control group, 4.15 for “example-only”, 1.53 for “tool-only”, and 2.71 for “tool and example” (see Figure 1). There was a reliable effect of group assignment on security scores ($F(3,173) = 6.05, MSE = 69.42, p < .01, \eta^2 = .09$), but there was insufficient evidence of a reliable effect of gender on security scores ($F(1,173) = .55, MSE = 6.35, p = .46, \eta^2 < .01$). Tukey’s HSD (Table 2) indicates significant differences only for “example-only” to the control (diff. of means = 2.68, $p < .01$) and “example-only” to “tool-only” (diff. of means = 2.58, $p < .01$). Cohen’s d for mean “example-only” security score vs. control group security score is .77, which indicates a medium effect size. Cohen’s d for “example-only” vs. “tool-only” is .73, also a medium effect size.

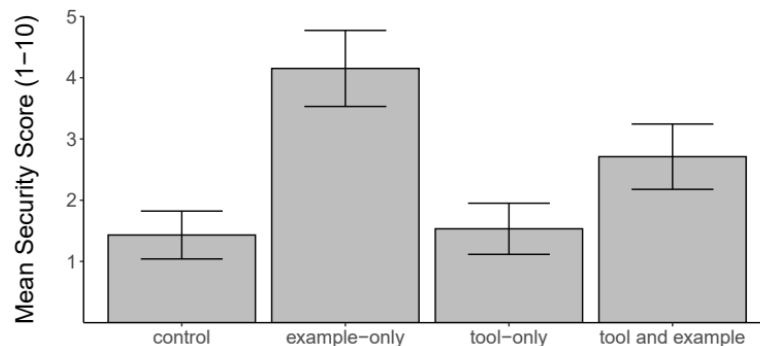


Figure 1. Mean security scores by study 1 group, with error bars depicting the standard error of the mean.

Table 2. Comparison of mean security scores by group, study 1 (Tukey HSD).

Comparison	Difference of Means	p
Example-only - Control	2.68	<.01
Tool-only - Control	.10	>.99
Tool and Example - Control	1.28	.29
Example-only - Tool-only	2.58	<.01
Example-only - Tool and Example	1.40	.21
Tool and Example - Tool-only	1.18	.35

Security scores were highly variable across all experimental conditions. A large number of students failed to make any significant progress towards a secure solution (103 out of the 180 students received zero security points).

Overall, security scores were quite low, and very few students completely secured the application against CSRF (0% in the control group, 24% in “example-only”, 0% in “tool-only”, and 13% in “tool and example”).

Despite common perceptions, prior work suggests that CS grades are typically unimodal (Patitsas et al., 2016). Consistent with past research, Hartigan’s dip test fails to reject the hypothesis that COMP 215 grades are unimodal ($D = .02, p = .93$). However, there is significant evidence of non-unimodality in the “example-only” security scores ($D = .12, p < .01$).

Functionality Scores. The mean functionality scores were .59 for the control group, .63 for “example-only”, .67 for “tool-only”, and .49 for “tool and example” (see Table 3). There was a reliable effect of gender on functionality ($F(1, 173) = 6.89, MSE = .55, p = .01, \eta^2 = .04$), but there was insufficient evidence of a reliable effect of group on functionality ($F(3, 173) = 1.31, MSE = .30, p = .27, \eta^2 = .02$). Table 4 shows the differences of mean functionality scores between the groups, none of which showed evidence of a reliable effect. For male vs. female functionality scores, difference of means = .21, $p = .01$, and Cohen’s d is .43, indicating a small effect.

Table 3. Functionality scores (0 or 1), study 1.

Group	<i>n</i>	Mean	<i>SD</i>
Control	44	.59	.50
Example-only	46	.63	.49
Tool-only	45	.67	.48
Tool and Example	45	.49	.51

Table 4. Comparison of mean functionality scores by group, study 1 (Tukey HSD).

Comparison	Difference of Means	<i>p</i>
Example-only - Control	.07	.91
Tool-only - Control	.08	.88
Tool and Example - Control	.10	.75
Example-only - Tool-only	.01	>.99
Example-only - Tool and Example	.17	.34
Tool and Example - Tool-only	.18	.30

Time on Task. Timing data is unavailable for a total of eleven students (two from the control group and three each from the other groups). There were three types of student error that led to missing or excluded timing data: failure to commit and push code before starting the task, large unexplained discrepancies between reported time and time measured via git pushes, and breaks taken without reporting the duration. The mean time on task in minutes (Table 5) was 155.7 for the control group ($n = 42, SD = 32.83$), 149.1 for “example-only” ($n = 43, SD = 34.05$), 151.3 for “tool-only” ($n = 42, SD = 41.21$), and 148.1 for “tool and example” ($n = 42, SD = 38.59$). We did not observe evidence of a reliable effect.

Table 5. Time on task, study 1.

Group	<i>n</i>	Mean	<i>SD</i>
Control	42	155.7	32.83
Example-only	43	149.1	34.05
Tool-only	42	151.3	41.21
Tool and Example	42	148.1	38.59

Multiple Linear Regression. To gain a better understanding of factors that may contribute to success in repairing CSRF vulnerabilities, we performed multiple linear regression on the security scores of each experimental condition, with time on task, final grade in COMP 215, GPA, number of the three security lectures attended, and self-reported Java experience (rated from 1-5) as possible factors (Table 4). In the control group, adjusted $R^2 = .24$, $F(5,30) = 3.24$, $DF = 30$, and $p = .02$. Final grade was the only significant factor. In “example-only”, adjusted $R^2 = .32$, $F(5,33) = 4.55$, $DF = 33$, and $p < .01$, and the only significant factor was time on task. In “tool-only”, adjusted $R^2 = .28$, $F(5,32) = 3.80$, $DF = 32$, and $p < .01$, and the significant factors were lectures attended and Java experience. Finally, in “tool and example”, adjusted $R^2 = .22$, $F(5,32) = 3.03$, $DF = 32$, and $p = .02$, and there were no significant factors.

Table 6. Multiple linear regression, security scores (“Tool” = tool-only, “Ex.” = example code only, “Both” = tool and example code).

FACTOR	η^2				<i>t</i>				<i>p</i>			
	Control	Ex.	Tool	Both	Control	Ex.	Tool	Both	Control	Ex.	Tool	Both
Time on Task	.05	.14	<.01	.05	1.41	.56	.09	1.4	.17	.02	.93	.15
Final Grade	.15	.07	.03	.07	2.48	1.8	1.0	1.6	.02	.08	.29	.10
GPA	<.01	<.01	.01	.02	.07	.03	.75	.88	.95	.98	.46	.39
Lectures Attended	.05	.02	.11	.03	1.43	.88	2.2	1.0	.16	.38	.03	.31
JAVA Experience	.05	.04	.15	.05	1.47	1.3	2.5	1.4	.15	.17	.01	.16
						7	9	5				

2.2.1 Post-task Survey Results

CSRF Testing Tools. SUS Scores In the “tool-only” group, the mean SUS score for CSRF Testing Tools was 42.94 ($n = 40$, $SD = 19.78$). In the “tool and example” group, it was 39.10 ($n = 39$, $SD = 19.97$). We did not observe a reliable effect of example code availability on SUS scores, based on Welch’s *t*-test ($DF = 77$, $t = .86$, $p = .39$). According to the adjective rating scale developed by Bangor et al. (Bangor et al., 2009), these scores fall between “poor” (mean score of 35.7) and “OK” (mean score of 50.9), indicating a very low degree of usability compared to other systems in general (though not necessarily to other CSRF mitigation tools). Moreover, it is rare in practice to find SUS scores below 40 for complex multi-step tasks, such as CSRF mitigation (Kortum and Acemyan, 2013).

Security and Functionality Confidence. We asked participants to rate their confidence in their solution in terms of security and functionality, each on a scale from 1 (low) to 5 (high). Note that self-reported task completion and confidence did not affect participants' participation credit in the course. The mean security confidence was 2.00 in the control group ($n = 37$, $SD = 1.11$), 2.82 in "example-only" ($n = 38$, $SD = 1.27$), 1.93 in "tool-only" ($n = 40$, $SD = 1.27$), and 2.03 in "tool and example" ($n = 39$, $SD = 1.20$). The mean functionality confidence was 2.65 in the control group ($n = 37$, $SD = 1.57$), 2.92 in "example-only" ($n = 38$, $SD = 1.65$), 2.38 in "tool-only" ($n = 40$, $SD = 1.53$), and 2.23 in "tool and example" ($n = 39$, $SD = 1.49$).

To measure how well students were able to gauge their own performance, we examined the correlations between confidence and actual scores, for both security and functionality. For security, the correlation was .59 in the control group ($t = 4.31$, $DF = 35$, $p < .01$), .60 in "example-only" ($t = 4.48$, $DF = 36$, $p < .01$), .50 in "tool-only" ($t = 3.57$, $DF = 38$, $p < .01$), and .68 in "tool and example" ($t = 5.66$, $DF = 37$, $p < .01$). For functionality, the correlation was .38 in the control group ($t = 2.45$, $DF = 35$, $p = .02$), .50 in "example-only" ($t = 3.45$, $DF = 36$, $p = .01$), .21 in "tool-only" ($t = 1.31$, $DF = 38$, $p = .20$), and .65 in "tool and example" ($t = 5.15$, $DF = 37$, $p < .01$). In summary, there was a significant and moderately large correlation between confidence and actual scores in all cases except for functionality in the "tool-only" group.

With regard to perceived security, arguably the most dangerous situation is one in which a developer believes strongly that they have protected the system against CSRF when they have not. In this situation, not only does the system remain vulnerable, but the developer is unlikely to seek help to repair the vulnerability. To quantify this condition, we define the "false confidence" rate as the proportion of students whose submission had a security score of zero for which (1) the student reported that they had completed the task and (2) the student rated their confidence in the security of the system as either a four or five on a 1-5 scale. The false confidence rates were .035 for the control group ($n = 28$), 0 for "example-only" ($n = 16$), .107 for "tool-only" ($n = 28$), and .053 for "tool and example" ($n = 19$). The false confidence rates are fairly low across the board, which is a reassuring result. We did not observe a reliable effect of experimental condition on the rate of false confidence.

3. Study 2: Impact of team programming

Study 2 examined the impact of working in pairs on the learning process. We recognized that the most successful condition ("example-only") from study 1 gave the students example code, but no CSRF tooling, so we used that as the starting point in study 2.

3.1 Methodology

We drew our students from the 2019 COMP 215 class at Rice University. The 2019 curriculum for this course was similar to that of 2018, with the notable exception of the introduction of partners for weekly projects, while the 2018 class projects were all performed solo. In 2019, beginning in week 7, students partnered with a different classmate of their choosing for each weekly project. The CSRF project occurred during week 11, after three such partnered projects, the same point in the semester as in study 1.

3.1.1 Design

Our goal for study 2 was to understand the impact of single versus two-person teams, so we used a between-subjects design with one variable: "Solo" vs. "Duo". We provided both groups access to example code and

neither group access to CSRF Testing Tools (i.e., the “example-only” condition). The dependent variables remain the same as in study 1, with the exception of SUS results for CSRF Testing Tools since it was unused.

Condition Assignment. Students were randomly assigned into one of the two conditions, subject to two constraints. First, consistent with COMP 215 team policy, no two students were assigned to work on a team who had previously worked together. Several COMP 215 students had special exemptions from this rule and were allowed to work with the same partner each week; these students completed the assignment but are excluded from the study results. Second, we asked any students who may have difficulty collaborating during the study (e.g., due to travel) to inform us so that they could be assigned to work alone. Twelve students out of a total of 163 informed us of such difficulties and were assigned to the Solo group. The remaining 151 students were split into the two groups uniformly at random. In total, the Solo group had 53 students, and the Duo group had 110. Note that there are approximately twice as many individuals in the Duo condition as in the Solo condition, in order to maintain an approximately equal numbers of teams.

For logistical reasons, it was not practical to keep it a secret from the students that some of them were working alone and some with a partner. Therefore, unlike in study 1, participants in study 2 were aware of their condition assignment. In the Solo group, student ages ranged from 18 to 21 years ($Mdn = 19$, $\bar{X} = 19.3$, $SD = .696$). Sixteen students identified as female and 36 as male, and one gave no response. Self-reported Java experience prior to taking COMP 215 ranged from 1 to 4 ($Mdn = 2$, $\bar{X} = 2.132$, $SD = .981$). In the Duo group, ages ranged from 18 to 26 years, ($\bar{X} = 19.4$, $Mdn = 19$, $SD = 1.160$). There were 28 female students, 81 male, and one identifying as neither male nor female. Self-reported Java experience ranged from 1 to 5 ($Mdn = 2$, $\bar{X} = 1.926$, $SD = .924$).

Timing. Unlike study 1, we did not examine git push times. This would have been infeasible for the Duo condition, where each student has a separate time-on-task and either student may push to the repository at any time. Instead, we simply asked each student to report their personal time on-task in minutes, working with or without their partner, excluding any breaks. We used the same timing reporting scheme for the Solo condition, for consistency.

3.2 Results and Discussion

Security Scores. The mean security score was 2.57 for Solo and 4.95 for Duo (Figure 2), and there was evidence of a reliable effect ($t = 2.94$, $DF = 105$, $p < .01$). Cohen’s $d = .56$, indicating a medium effect size. As in study 1, security scores were variable but generally low, with 52 of 108 students receiving zero security points. Only 25% of Duo teams and 17% of solo teams fully secured the application against CSRF. Hartigan’s dip test indicates non-unimodality in both Solo ($\mathcal{D} = .087$, $p < .01$) and Duo ($\mathcal{D} = .161$, $p < .01$) security scores. As in 2018, the 2019 COMP 215 final grades exhibit no reliable non-unimodality ($\mathcal{D} = .025$, $p = .70$). There was a reliable effect of group assignment ($F(1,101) = 9.47$, $MSE = 161.07$, $p < .01$) but not of team gender composition ($F(3,101) = 1.91$, $MSE = 32.56$, $p = .13$) on security scores. We also note that despite the “example-only” group from study 1 and the Solo group from study 2 having similar experimental conditions, “example-only” had a mean security score of 4.15 vs. 2.56 for the Solo group. However, despite the apparently worse performance of the Solo group, there was insufficient evidence of a reliable effect ($t = 1.90$, $DF = 94$, $p = .06$), and the effect size is small (Cohen’s $d = .38$).

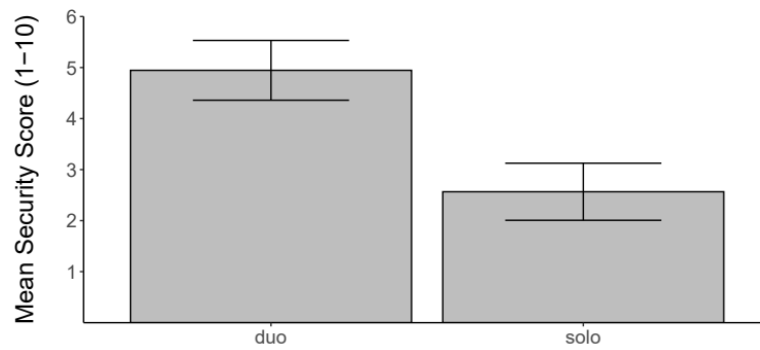


Figure 2. Mean security scores by study 2 group, with error bars depicting the standard error of the mean.

Functionality Scores. The mean functionality scores (Table 7) were .62 for Solo and .67 for Duo. There was insufficient evidence of a reliable effect on functionality scores of either team gender composition ($F(3,101) = 1.80$, $MSE = .41$, $p = .15$) or group ($F(1,101) = .57$, $MSE = .13$, $p = .45$).

Table 7. Functionality scores (0 or 1), study 2.

Group	<i>n</i>	Mean	<i>SD</i>
Solo	53	.62	.49
Duo	55	.67	.47

Time on Task. Timing data is unavailable for a total of six teams (four from Solo and two from Duo), due to students neglecting to complete the post-task survey. When only one student in a Duo team reported the time on task, we take that figure as the team average. The mean time on task in minutes (Table 8) was 181 for Solo ($n = 49$, $SD = 37.6$) and 169 for Duo ($n = 53$, $SD = 39.8$).

Table 8. Time on task, study 2.

Group	<i>n</i>	Mean	<i>SD</i>
Solo	49	181	37.6
Duo	53	169	39.8

Multiple Linear Regression. We performed multiple linear regression once again to try to determine which factors we examined contribute most to success in repairing CSRF vulnerabilities. As before, we used time on task, final grade in COMP 215, GPA, number of the three security lectures attended, and self-reported Java experience (rated from 1-5) as factors (Table 9). For the Duo condition, since there are two students per submission, we used the team average for each factor. In Solo, adjusted $R^2 = .32$, $F(5,40) = 5.24$, $DF = 40$, and $p < .01$. The significant factors were time on task, final grade, and lectures attended. In Duo, adjusted $R^2 = .061$, $F(5,47) = 1.67$, $DF = 47$, and $p = .16$, and no factors showed evidence of a reliable effect.

Table 9. Multiple linear regression, Solo and Duo security scores.

FACTOR	η^2		<i>t</i>		<i>p</i>	
	Solo	Duo	Solo	Duo	Solo	Duo

Time on Task	.09	.04	2.40	1.47	.02	.15
Final Grade	.17	.04	3.30	1.53	<.01	.13
GPA	.02	<.01	1.22	.44	.23	.67
Lectures Attended	.09	.01	2.34	.69	.02	.49
JAVA Experience	.01	.02	.63	1.13	.53	.26

3.2.1 Post-task Survey Results

Security and Functionality Confidence. We again asked students to rate their security and functionality confidence from 1-5. The mean security confidence was 2.25 (SD 1.15) in Solo and 2.91 (SD1.18) in Duo. The mean functionality confidence was 2.71 (SD 1.53) in Solo versus 3.19 (SD 1.41) in Duo.

Tables 10 and 11 show the security and functionality confidence correlations for study 2. For security, the correlation was .60 in Solo and .33 in Duo. For functionality, the correlation was .29 in Solo and .57 in Duo. There was a significant and moderately large positive correlation between confidence and actual scores in all cases in study 2. Using the same definition of “false confidence” as in study 1, the rates in study 2 were 0 for Solo ($n = 35$) and .147 for Duo ($n = 34$). Five individuals from three different Duo groups exhibited false confidence, whereas no Solo students did. This time, experimental condition did have a reliable effect on the false confidence rate ($F(1,61) = 5.86, MSE = .41, p = .02, \eta^2 = .09$). Cohen’s d is .58, a medium effect size. Interestingly, in exactly one team with a security score of zero, there was a mismatch of confidence levels, with one partner highly confident (4) and one not (zero).

Table 10. Confidence-security correlation, study 2.

Group	Pearson’s r	t	DF	p
Solo	.60	5.20	47	<.01
Duo	.33	3.46	97	<.01

Table 11. Confidence-functionality correlation, study 2.

Group	Pearson’s r	t	DF	p
Solo	.29	2.10	47	.04
Duo	.57	6.82	97	<.01

We were surprised to see that the Duo group had a higher false confidence rate. One potential explanation is that participants working in teams tended to rely on and trust their partners, assuming the teammate knew more than they actually did. This result may be cause for some concern and caution when working collaboratively, but in terms of security results, the Duo teams still performed much better overall.

4. General Discussion

Our results identify important factors for improving student success rates in learning how to repair CSRF vulnerabilities, as well as some factors that surprisingly had little or no effect. These results also include a baseline SUS score for any future usability studies on CSRF prevention tools, which is valuable for drawing standard and objective comparisons between the satisfaction such tools provide users.

Consistent with our expectations and with previous research (see, e.g., Acar et al., 2017), access to reliable example code had a significant impact on the security of students' solutions, at least when not coupled with CSRF Testing Tools. Students in the "example-only" group had moderately higher security scores than both the control group and the group with access to the tool alone. These results confirm the efficacy of quality example code in training students to improve code security. Also in line with expectations, student teams of two produced significantly more secure solutions than students working alone. To the best of our knowledge, this study is the first to empirically demonstrate that working in pairs can result in better outcomes when training for computer security tasks.

Contrary to our expectations, access to CSRF Testing Tools did not reliably improve security scores. Even more surprisingly, *the tool actually appears to have hurt the security scores for students with example code*. We can only postulate what went wrong. One possibility is that the students were constrained by the time limit, and the tool distracted students from spending time understanding and adapting the example code. We know that in at least some cases, the tool actively misled students into thinking an insecure solution was secure. One student commented in a git commit message that they were done because "[the page] runs on FormBuilder"; i.e., the spoofed form generated by the tool worked, and the student interpreted that as an indication that the server was secure even though it actually implies the exact opposite. This is an easy mistake to make for a developer lacking computer security expertise when using a tool with little documentation. Furthermore, among the 79 students with access to CSRF Testing Tools who completed the post-task survey, nine (11.4%) specifically mentioned understanding the tool as one of the most difficult parts of the assignment.

In absolute terms, the scores we obtained in study 1 indicate that CSRF Testing Tools is not perceived to be usable by the students. However, we cannot conclude that CSRF Testing Tools is necessarily worse in this regard than other CSRF prevention tools, since this is the first study to apply the SUS to this class of tools. However, researchers and tool authors can compare the satisfaction of existing and future products against this baseline freeware tool, providing empirical evidence for their comparative usability. Anecdotally, while a few students commented that the tool was helpful in solving the task, many other students indicated that they did not understand how to begin using the tool or how to interpret its behavior. We conjecture that more detailed documentation, including a description of what to expect when the CSRF vulnerability has or has not been patched, could improve perceived usability.

Students sometimes erroneously think their code is secure when it is not. In the field, this false sense of security can result in security defects making their way into deployed software, putting end users at risk. Previous studies have found differing results regarding the correlation between how secure developers think their code is and how secure it actually is (Acar et al., 2017; Gorski et al., 2018). In this study, we found that students' perceptions of their code's functional correctness and security generally matched their actual performance. It is possible that these correlations were driven by the appreciable number of students who failed to make any significant progress towards a solution and in that case knew for certain that they had failed. We were surprised to see that the Duo

group had a higher false confidence rate. One potential explanation is that participants working in teams tended to rely on and trust their partners, assuming the teammate knew more than they actually did. This result may be cause for some concern and caution when working collaboratively, but in terms of security results, the Duo teams still performed much better overall.

We expected that students with higher GPAs and/or final grades in the course would produce more correct and more secure solutions. However, GPA did not have a significant effect, and final grades in COMP 215 were only predictive in the study 1 control and study 2 Solo groups. We also expected prior experience with Java to have a positive impact on security outcomes, but we only observed a significant effect of self-reported Java experience on security scores in the “tool-only” group.

4.2 Limitations

We limited our students’ time on task to 180 minutes. The primary motivation for restricting the time on task was to prevent students from wasting too much time on the assignment if they became stuck. A secondary motivation was to allow us to encourage the students to complete the task in a single contiguous block of time, reducing errors in recorded times (e.g., if students failed to report breaks). Unfortunately, since many students used the entire allotted time, the time restriction makes it difficult to distinguish between students who truly became stuck and students who could have made additional significant progress if given more time. One possible mitigation would be to have a prior week’s assignment using the exact same codebase, only without security considerations, and thus give the students additional familiarity with the experimental setup. We could also try to further simplify the security task, although this particular task is already quite simple, at least for students who understand the problem.

Lecture attendance had a *negative* correlation with security outcomes in every group except “example-only”, and the effect was significant in the “tool-only” and Solo groups. This effect is inconsistent, and one possible explanation is that stronger students might not bother attending lectures, in general. However, it is also entirely possible that our security lectures did not do a good job of explaining CSRF, and that many students solved the problem simply by transferring the coding pattern from the example without understanding how it works or why it is important. Prior research has shown that copying and pasting code can lead to security bugs (Fischer et al., 2017), underscoring the importance of reliable code samples but also of security awareness and critical thinking on the part of developers. A future study might provide more insight into students’ security comprehension, in addition to security performance, by including a series of security questions before and after the assignment, to gauge students’ security skills “on paper”.

5. Conclusions and Future Work

In this work, we conducted two large studies with undergraduate computer science students to investigate which factors affect students’ ability to successfully complete a computer security assignment, viz. securing a web app against cross-site request forgery. In study 1, we examined the impact of providing students with particular resources during the assignment: a fuzz-testing tool and/or example code. In study 2, we looked at the effects of working alone vs. in a pair.

We found that providing students with both example code and a software tool does not appear to confer the additive benefit to learning outcomes we expected. Our results establish the benefit of access to example code when attempting to teach CSRF prevention but also demonstrate that an unusable tool can not only fail to significantly improve learning how to implement good security outcomes but actually be detrimental to success in some cases.

The results of our follow-up study further demonstrate that working with a partner has substantial benefits to learning outcomes when compared with working alone. Prior work both in education and industry has demonstrated that collaborative programming increases confidence and satisfaction, reduces frustration, improves course and major retention, and otherwise improves outcomes (Williams and Upchurch, 2001; Williams et al., 2002; McDowell et al., 2002; McDowell et al., 2003; Werner et al., 2004; McDowell et al., 2006; Eierman and Iversen, 2018; Williams et al., 2000; Williams et al., 2002; Hanks et al., 2004; Smith et al., 2018; Werner et al., 2004; McDowell et al., 2006; Arisholm et al., 2007; Begel and Nagappan, 2008; Hannay et al., 2009). As such, it's unsurprising but important to note that working in pairs also improves educational outcomes in training students in security measures.

Based on these results, we suggest three key recommendations for instructors with regard to teaching students how to implement secure code: (1) seek out trustworthy example code to give to students for applying security coding techniques, (2) be wary of tools in the classroom whose efficacy is unverified, and (3) when possible, allow students to collaborate in solving these kinds of security challenges, rather than working alone. Although we used CSRF prevention as our example, it's quite likely that these findings apply to teaching many other classes of security challenges being taught in the classroom. In sum, example code helps. Partners help. Bad tools hurt.

References

- Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., Mazurek, M. L., and Stransky, C. (2017). Comparing the Usability of Cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 154-171. <https://doi.org/10.1109/SP.2017.52>
- Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M. L., and Stransky, C. (2016). You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289-305. <https://doi.org/10.1109/SP.2016.25>
- Acar, Y., Stransky, C., Wermke, D., Mazurek, M. L., and Fahl, S. (2017). Security Developer Studies with GitHub Users: Exploring a Convenience Sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 81-95, Santa Clara, CA. USENIX Association. <https://www.usenix.org/system/files/conference/soups2017/soups2017-acar.pdf>
- Arisholm, E., Gallis, H., Dyba, T., and Sjoberg, D. I. K. (2007). Evaluating pair programming with respect to system complexity and programmer expertise. *IEEE Transactions on Software Engineering*, 33(2):65-86. <https://doi.org/10.1109/TSE.2007.17>
- Bangor, A., Kortum, P., and Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of Usability Studies*, 4(3):114-123. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.177.1240&rep=rep1&type=pdf>
- Bangor, A., Kortum, P. T., and Miller, J. T. (2008). An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 24(6):574-594. <https://doi.org/10.1080/10447310802205776>

- Begel, A. and Nagappan, N. (2008). Pair programming: What's in it for me? In *Proceedings of the Second ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '08*, page 120-128, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/1414004.1414026>
- Bernhard, M., Sharman, J., Acemyan, C. Z., Kortum, P., Wallach, D. S., and Halderman, J. A. (2019). On the Usability of HTTPS Deployment. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI '19*, pages 310:1-310:10, New York, NY, USA. ACM. <https://doi.org/10.1145/3290605.3300540>
- Brooke, J. (1996). SUS: A quick and dirty usability scale. Usability evaluation in industry, page 189.
- Eierman, M. and Iversen, J. (2018). Comparing testdriven development and pair programming to improve the learning of programming languages. *Journal of the Midwest Association for Information Systems*, 2018:23-36. <https://doi.org/10.17705/3jmwa.000038>
- Fischer, F., Böttinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., and Fahl, S. (2017). Stack Overflow Considered Harmful? The Impact of Copy Paste on Android Application Security. In *2017 IEEE Symposium on Security and Privacy*, pages 121-136. <https://doi.org/10.1109/SP.2017.31>
- Gorski, P. L., Iacono, L. L., Wermke, D., Stransky, C., Moeller, S., Acar, Y., and Fahl, S. (2018). Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse. In *SOUPS @ USENIX Security Symposium*. <https://www.usenix.org/system/files/conference/soups2018/soups2018-gorski.pdf>
- Hanks, B., McDowell, C., Draper, D., and Krnjajic, M. (2004). Program quality with pair programming in cs1. In *ACM Sigcse Bulletin*, (36), pages 176-180. <https://doi.org/10.1145/1026487.1008043>
- Hannay, J., Dybå, T., Arisholm, E., and Sjøberg, D. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51:1110-1122. <https://doi.org/10.1016/j.infsof.2009.02.001>
- ISO 9241-11 (2018). *Ergonomics of human-system interaction - Part 11: Usability: Definitions and concepts*.
- Kortum, P. and Acemyan, C. Z. (2013). How low can you go?: Is the system usability scale range restricted? *Journal of Usability Studies*, 9(1):14-24. http://tecfa.unige.ch/tecfa/maltt/ergo/1415/UtopiaPeriode4/articles/JUS_Kortum_November_2013.pdf
- Krombholz, K., Mayer, W., Schmiedecker, M., and Weippl, E. (2017). 'I Have No Idea What I'm Doing' - On the Usability of Deploying HTTPS. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1339-1356, Vancouver, BC. USENIX Association. <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-krombholz.pdf>

- McDowell, C., Werner, L., Bullock, H., and Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. In *ACM SIGCSE Bulletin*, (34), pages 38-42. <https://doi.org/10.1145/563340.563353>
- McDowell, C., Werner, L., Bullock, H. E., and Fernald, J. (2003). The impact of pair programming on student performance, perception and persistence. In *25th International Conference on Software Engineering*, 2003, pages 602-607. <https://doi.org/10.1109/ICSE.2003.1201243>
- McDowell, C., Werner, L., Bullock, H. E., and Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, 49(8):90-95. <https://doi.org/10.1145/1145287.1145293>
- Mindermann, K. and Wagner, S. (2018). Usability and Security Effects of Code Examples on Crypto APIs - CryptoExamples: A platform for free, minimal, complete and secure crypto examples. *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. <https://doi.org/10.1109/PST.2018.8514203>
- Mindermann, K. and Wagner, S. (2020). Fluid Intelligence Doesn't Matter! Effects of Code Examples on the Usability of Crypto APIs. *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 306-307. <https://doi.org/10.1145/3377812.3390892>
- Naiakshina, A., Danilova, A., Gerlitz, E., and Smith, M. (2020). On conducting security developer studies with cs students: Examining a password-storage study with cs students, freelancers, and company developers. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1-13. <https://doi.org/10.1145/3313831.3376791>
- Naiakshina, A., Danilova, A., Gerlitz, E., von Zezschwitz, E., and Smith, M. (2019). 'If You Want, I Can Store the Encrypted Password': A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI '19*, pages 140:1-140:12, New York, NY, USA. ACM. <https://doi.org/10.1145/3290605.3300370>
- Naiakshina, A., Danilova, A., Tiefenau, C., and Smith, M. (2018). Deception task design in developer password studies: Exploring a student sample. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 297-313, Baltimore, MD. USENIX Association. <https://www.usenix.org/system/files/conference/soups2018/soups2018-naiakshina.pdf>
- Oliveira, D. S., Lin, T., Rahman, M. S., Akefirad, R., Ellis, D., Perez, E., Bobhate, R., DeLong, L. A., Cappos, J., and Brun, Y. (2018). API Blindspots: Why Experienced Developers Write Vulnerable Code. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 315- 328, Baltimore, MD. USENIX Association. <https://www.usenix.org/system/files/conference/soups2018/soups2018-oliveira.pdf>

- Patitsas, E., Berlin, J., Craig, M., and Easterbrook, S. (2016). Evidence that computer science grades are not bimodal. In ICER '16: Proceedings of the 2016 ACM Conference on International Computing Education Research, pages 113-121. <https://doi.org/10.1145/2960310.2960312>
- Smith, M. O., Giugliano, A., and DeOrio, A. (2018). Long term effects of pair programming. *IEEE Transactions on Education*, 61(3):187-194. <https://doi.org/10.1109/TE.2017.2773024>
- Ukrop, M. and Matyas, V. (2018). Why Johnny the Developer Can't Work with Public Key Certificates - An Experimental Study of OpenSSL Usability. In CT-RSA, volume 10808 of *Lecture Notes in Computer Science*, pages 45-64. Springer. https://doi.org/10.1007/978-3-319-76953-0_3
- Weber, S., Coblenz, M., Myers, B., Aldrich, J., and Sunshine, J. (2017). Empirical studies on the security and usability impact of immutability. In 2017 *IEEE Cybersecurity Development (SecDev)*, pages 50-53. <https://doi.org/10.1109/SecDev.2017.21>
- Werner, L., Hanks, B., and McDowell, C. (2004). Pair programming helps female computer science students. *ACM Journal of Educational Resources in Computing*, 4. <https://doi.org/10.1145/1060071.1060075>
- Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software*, 17(4):19-25. <https://doi.org/10.1109/52.854064>
- Williams, L. and Upchurch, R. (2001). In support of student pair-programming. In *ACM SIGCSE Bulletin*, (33), pages 327-331. <https://doi.org/10.1145/366413.364614>
- Williams, L., Wiebe, E., Yang, K., Ferzli, M., and Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, 12(3):197-212. <https://doi.org/10.1076/csed.12.3.197.8618>
- Zeier, A., Wiesmaier, A., and Heinemann, A. (2019). *API Usability of Stateful Signature Schemes*, pages 221-240. Springer. https://doi.org/10.1007/978-3-030-26834-3_13

