A Case Study of Middle Schoolers' Use of Computational Thinking Concepts and Practices during Coded Music Composition

Yifan Zhang
Computer and Information Sciences
University of Delaware
ericzh@udel.edu

Douglas Lusa Krug
Computer Science
Virginia Commonwealth University
Instituto Federal do Paraná - IFPR
krugdl@vcu.edu

Chrystalla Mouza
School of Education
University of Delaware
cmouza@udel.edu

David C. Shepherd
Computer Science
Virginia Commonwealth University
shepherdd@vcu.edu

Lori Pollock
Computer and Information Sciences
University of Delaware
pollock@udel.edu

ABSTRACT

Researchers and practitioners have demonstrated various benefits of introducing computational thinking (CT) through music composition coding. While researchers have studied the impacts on participant attitudes towards CT and their learning of CT concepts, more case studies are needed on both learning CT concepts as well as CT practices, i.e., the processes of constructing music coding projects. This paper presents a case study of middle schoolers in an informal learning environment focused on integrating music composition with coding in TunePad. Specifically, we collected and analyzed logs of coding events, final code products, and surveys to explore both CT concept use and CT practices exhibited by the participants as they completed open-ended music coding activities to create their own melodies with specific music and CT requirements and recommendations.

CCS CONCEPTS

• **Social and professional topics** → **K-12 education**; **Computational thinking**; **Informal education**; • **Applied computing** → **Sound and music computing**.

KEYWORDS

computational thinking (CT), music composition, informal learning

ACM Reference Format:
Yifan Zhang, Douglas Lusa Krug, Chrystalla Mouza, David C. Shepherd, and Lori Pollock. 2022. A Case Study of Middle Schoolers' Use of Computational Thinking Concepts and Practices during Coded Music Composition. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol 1 (ITiCSE 2022), July 8–13, 2022, Dublin, Ireland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3502718.3524757

ITiCSE 2022, July 8–13, 2022, Dublin, Ireland.

ACM ISBN 978-1-4503-9201-3/22/07…$15.00
https://doi.org/10.1145/3502718.3524757

1 INTRODUCTION

Since Wing's [28] seminal paper, Computational Thinking (CT) has been increasingly studied, especially in K-12 education [16]. Barr and Stephenson defined CT as a problem-solving methodology that can be automated, transferred, and applied across disciplines [4]. With the challenges of fitting CT into already full curricula, teachers are interested in ways of integrating CT into the learning of other subjects. By integrating CT with content, students avoid seeing CT in isolation, class time is used efficiently, and students who might not see themselves attracted to CT may find it more naturally applicable to them [5, 20].

While CT is often viewed as machine-centric in the context of computer science, the integration of music and CT is particularly promising as music is an art encouraging creativity, communication and teamwork. At the same time, music and CT share several commonalities in the use of notation, sequence, repetition, and creativity [5]. Several researchers including Bell and Bell [5], Baratè et al. [3], and Petrie [24] examined meaningful ways of integrating CT with music learning and studied students' learning outcomes in relation to CT concepts such as decomposition, pattern recognition, abstraction, and algorithmic thinking. Their results demonstrate multiple benefits as a result of integrating music learning and CT.

The above studies, however, mostly focused on the learning of CT concepts alone. Yet Brennan and Resnick [6], as well as other researchers such as Horst et al. [15], Zhang and Nouri [29], and Allsop [2], strongly recommend that in addition to examining learning of CT concepts, research in this area should also examine learning of CT practices. CT practices are the processes students follow to construct their projects, that is, the design practices that students are using as they think and learn. Examples of such practices, include: (a) being incremental and iterative, (b) testing and debugging, (c) reusing and remixing, and (d) abstracting and modularizing [6]. To our knowledge, research focusing on how students develop CT practices in addition to CT concepts during integrated CT and music learning is lacking from the literature.

In this paper, we describe a case study focusing on assessing students' CT concepts and practices in the context of CT-integrated music. The work was conducted in the context of an informal learning environment where participants composed music through

coding in TunePad, a free, online platform for creating music using the Python programming language [14]. Specifically, we examine middle schoolers (grades 5-8) CT concepts and practices through their participation in a two-week online summer camp. We analyzed the participants' coding process logs to explore their CT practices along with the participants' coding products to identify the CT concepts used to meet the requirements and recommendations of daily camp activities.

To explore CT practices, we logged participants' events during their coding and used a framework by Brennan and Resnick to analyze the process logs. The example CT practices proposed by Brennan and Resnick are similar to tinkering behaviors in coding. Specifically, in the context of block-based coding environments, Dong et al. [8] defined tinkering as the process by which students engage with testing, debugging, and struggling with code in order to achieve a goal particularly in an open-ended programming assignment. Unlike most introductions to programming where students are coding to create a well-specified outcome, activities in our summer camp were all open-ended with specific requirements and recommendations, but no correct melody that all students were aiming to create. Given the freedom to code creatively through music composition, we sought to use process logs to identify tinkering. We also developed an assessment metric for CT concepts that captures completion of requirements and recommendations in the context of the camp activities.

Specifically, we designed our study to answer the following research questions:

RQ1: What CT concepts are evident in participants' coding products? What percentage of participants met the requirements and recommendations of daily activities?

RQ2: What CT practices, in the form of tinkering behaviors, were exhibited in participants' process logs during music coding in daily activities?

RQ3: How did participants' tinkering behaviors during daily tasks compare with their tinkering behaviors during the final competition capstone task?

2 RELATED WORK

Related research can be categorized in three strands: coding environments for music composition, approaches to integrating music and CT learning, and research focusing on student learning when integrating CT and music.

The major music coding environments currently used include EarSketch [22], TunePad [14], Sonic Pi [1], and general block-based programming environments such as Scratch that also support music coding [26]. EarSketch, developed by Magerko et al. primarily for introductory computer science, allows users to remix music by manipulating audio samples using the Python programming language, which is commonly used to teach computer science at the introductory level [22]. More recently, Horn et al. developed an interactive web-based environment called TunePad, which is built on EarSketch and utilize the Python programming language [14]. Sonic Pi, developed by Aaron et al., enables music coding through a language based on the Ruby programming language. Sonic Pi's unique feature is its ability to support live coding where the user can modify the code while it is being played, continuing to generate music [1]. Unlike these environments that are designed specifically for music coding, some block-based languages can be used to create music. Greher and Heines are particularly credited with showing how Scratch can be used for music coding [12].

To date, there has been work on developing music and CT integrated curricula in various contexts. An early, notable integration of music and CT was the media computation course developed by Guzdial for introducing computation to university non-majors [13]. Focusing on K-12 students, Bell and Bell explored and tested creative ways to connect CT and music, including parallel sorting through comparing musical elements such as note pitches, exploring binary representations using music, and coding music in Scratch [5]. Similarly, Baratè et al. developed coding exercises for middle school students by customizing the Blockly block-based environment to enable creation of melodies [3]. Recently, Krug et al. presented Code Beats, an approach that teaches middle school students to program using hip hop beats, intentionally leveraging a genre of music that appeals to a wide array of urban youth of color using Sonic Pi [21].

To examine the impacts of CT-integrated music approaches on participants, researchers have used surveys, interviews, focus groups, and final coding products. For instance, several studies have shown that EarSketch coding can promote student engagement [9, 11, 17], creativity [9, 11], and other affective outcomes [23] in high school classrooms. Similarly, Koppe reported that a diverse group of participants coding with Sonic Pi in a workshop developed medium to highly complex programs and demonstrated positive attitude towards their experience [18]. In a team-based learning context using Sonic Pi, Traversaro et al.'s results showed better student performance and reduced course dropout compared to individual learning and positive student attitudes [27]. Further, Burnard et al.'s work demonstrated that music and CT integration contributed to increases in confidence in both subjects [7]. Finally, through a case study using pre and post semi-structured interviews, class quizzes, and student reflections, Petrie found that middle schoolers participating in integrated music coding with Sonic Pi demonstrated learning of CT concepts, practices, and perspectives [24].

While these studies are useful, they all focus on students' learning as evident in final products. To our knowledge, no studies exist that analyze process logs to gain a better understanding of CT practices.

3 CONTEXT OF OUR CASE STUDY

This section describes the informal learning structure, participants, music coding environment (i.e., TunePad), and open-ended music coding activities.

Informal Learning Structure. The context of our study is a 2-week (10 days) summer camp where each day consisted of a 1-hour online streamed learning session followed by an after-class programming assignment to reinforce the music and coding concepts taught that day. The learning session included introduction of computing concepts by computer scientists, music theory by a music expert, a worked music coding example in TunePad with live coding, highly scaffolded in-class coding activities, and a quiz competition. We held a 1-hour office hour immediately after the live streaming

for participants who wanted individual help. The participants were strongly encouraged to upload their music coding products each day for review.

For each daily after-class coding activity, participants were provided initial code sequences and instructions, which included both requirements to complete the activity and recommended commands to try in their coding. In this paper, we refer to these as the activity requirements and recommendations. In all cases, the activities were open-ended, leaving room for creativity and multiple potential solutions to complete the requirements. The last camp activity was a capstone project competition, where participants chose among three different initial music codes and added at least one original track through coding to create their own melody.

Participants. We held two instantiations of the 2-week summer camp to accommodate student interest. In total, 195 middle schoolers participated in at least one of the camps. Of those, 132 students agreed to participate in this research through parent signed consent and student assent. In the rest of this paper, we refer to the 132 who signed the consent form as the *participants*. However, it is important to note that not all 132 students completed all camp activities. This research is approved by the Institutional Review Board (IRB).

Music Coding Environment. With easy accessibility through a web browser from any platform, TunePad was used as our music coding environment [14]. The interface consists of a set of playable musical instruments at the top, a music timeline that shows the timing of different tracks with respect to each other, and an editing panel where Python code is written to create coded music.

Open-ended Music Composition Coding Tasks. In this paper, we focus on analyzing the data from the daily after-class activities and the capstone project, which were more open-ended than the short in-class coding activities. Due to the volume of data and space constraints, we selected two representative daily assignment tasks for this case study. Day 5, which covered the use of lists in music coding, was the last day of the first week of camp, midway through the camp. Day 8, which covered repetition (iteration) in music coding, was the last day of after-class activities before participants worked on their individual capstone projects. Lists and repetition as well as sequences, which are included in both of these days, are important overlapping concepts in both CT and music.

Table 1 presents the coding, music, and CT concepts covered during the instructional sessions followed by the after-class activity instructions in terms of task requirements and recommendations. On day 5, the music expert described chords while the computer scientist demonstrated lists in music coding, building on concepts students had already seen, including sequence, parallelism, and data. In the after-class activity, participants were given a song with 8 background instruments and asked to create 2 instruments (Chords - Intro and Chords - Verse) using commands *playNote* with available chords. In both instruments, we predefined 7 notes (i.e., C = 60, D = 50, etc) and 1 chords (i.e., F_major = [F, A, C]). We also predefined 4 empty chords and asked participants to fill in proper notes in each chords (i.e., E_min = [], D_min = [], etc). We provided several measures with code comments and three functions with different beat lengths as the basic structure of the instruments. Each measure was required to contain 4 beats, and recommended chords were listed as code comments (see Figure 1).

Table 1: Music coding tasks: concepts, requirements, and recommendations

Category	Items	Day 5	Day 8
Coding Concepts	Lists	√	
	Repetition with nested lists		√
Music Concepts	Chords	√	
	Chord progression		√
CT Concepts	Sequence	√	√
	Parallelism	√	√
	Data	√	√
	Loop		√
Task Requirements	Define 4 new chords	√	
	Use given chords		√
	Each measure has 4 beats	√	√
Task Recommendations	Play recommended chords for each measure	√	
	Use *for* with *lists* to create a melody		√

On day 8, the instructional session focused on musical chord progression and coding repetition with nested lists. Similar to day 5, participants were given 6 background instruments and asked to create 2 instruments. We asked participants to use the initial chords to compose their melody and include 4 beats with each measure. We also recommended that they use the *for* command with *lists* to create their melody.

```
1  F_maj = [F, A, C]
2  # Measure 1-2
3  # Try using chords F major, E minor and D minor
4  playNote(F_maj, 2)
```

Figure 1: Example initial Python code

For the final capstone project competition, we offered 3 initial codes for students to choose from as the basic structure of their competition song. These 3 initial sketches were identical in the types of instruments and number of tracks; the only difference was the beats length.

4 STUDY METHODOLOGY

Table 2 presents our data collection and analysis methods for our three research questions. We collected and analyzed three types of data: coding process logs, coding products, and participant survey data. Of the 31 and 22 middle schoolers who participated in the day 5 and day 8 instructional sessions respectively, 22 and 16 of them actually engaged in editing code for the after-class activities for those days, respectively. We believe the high attrition rate is due to the online camp which provided no supervision of after camp activities. We collected and analyzed data for the 22 and 16 participants from day 5 and day 8, respectively. While 51 middle schoolers participated in the final capstone project competition, we

studied the subset who also edited code for the after-class activity of either day 5 or day 8 activities, which was 14 participants.

Table 2: Questions, data sources, and methodology

Research Questions	Data Source	Analysis Method
RQ1	Coding product	Assessment metric measuring task requirements and recommendations completed
RQ2, RQ3	Process log data Survey data	Measurement for tinkering behavior and link to survey data

4.1 Data Collection

We collaborated with the TunePad developers to log user events as each participant in our camp (who provided consent) coded within the TunePad environment throughout the 2-week camp. To identify potential events to log, we started with the log data structure of the ProgSnap2 standard [25] and adapted it appropriately for music coding. We logged both coding process and code editing events. Coding process events include *edit-instrument, error-instrument, play-instrument,* and *play-project*. Play events in music coding are analogous to run/execute code events in general programming. Code editing event granularity is line-based and is logged when users move their input cursor to a new line.

In total, we logged 138,735 events over all participants throughout the two 2-week camps from the first day to the end of capstone project. For our case study days 5, 8, and capstone project, we logged 2,260, 1,459, and 21,110 events respectively.

As we collected event data, we also collected associated code snapshots (i.e., code at the time of that event). This enabled us to examine the state of the code at the time different events were taking place. We designate the final code product for each activity to be the final code snapshot for that activity's event log.

The pre-camp survey provided participant demographics and their prior music/coding knowledge/interests. This enabled us to situate our data. For day 5 (n=22), 11 identified as boys, 8 as girls, and 3 preferred not to identify; 12 indicated that they were interested in computing while 10 indicated they were interested in music. For day 8 (n=16), 6 identified as boys, 7 as girls, and 3 preferred not to identify; 8 were interested in computing and 6 were interested in music. For capstone (n=14), 9 identified as boys, 4 as girls, 1 preferred not to identify; 10 were interested in computing and 9 were interested in music.

4.2 Data Analysis

All data analysis was conducted by researchers not involved in camp instructions. For RQ1, which examines how participants met the requirements and followed recommendations, we organized the requirements and recommendations for each day's after-class activity by music and coding concepts, as presented in Table 3. We manually examined each participant's final code product to determine whether they met each requirement in the chart and followed each recommendation. We maintained a count of the number of participants who edited their code and met each requirement and recommendation, and computed percentages based on the students who edited code (n=22 on day 5 and n=16 on day 8). In Table 3, R represents requirement and C represents recommendation. Unlabeled descriptions were features that appeared in some codes as indicated by the counts but were not requirements or recommendations. For example, On day 5, 16 of 22 participants defined the required chords, and 6 of 22 participants played the recommended chords.

For RQ2, which attempts to analyze tinkering behavior during music coding, we focused on the coding process log data. Krieger et al. [19] conceptualized tinkering behaviors into four categories, which include exploratory behavior, deviation from instructions, lack of reliance on formal instruction, and use of trial and error. In this study, we focused on studying potential exploratory behavior and use of trial and error. For exploratory behavior, we adapted Dong et al.'s construction-based tinkering [8], which they define as *"a behavior where students make major changes in the code, like adding, deleting, or rearranging blocks... Construction-based tinkering behavior indicates that a student is likely pursuing an idea on how to solve the exercise, but still demonstrates some hesitation or uncertainty. We differentiate construction-based tinkering from non-tinkering construction behavior based on the amount of hesitation or uncertainty exhibited."*

Specifically, we analyzed two kinds of code changes: programming language token changes and added or deleted lines of code in TunePad. Similar to Dong et al. [8], we related more changes with more tinkering. We developed a parser to parse participants' coded music into several tokens, such as list, function, parameter, and loop. We then compared code snapshots associated with consecutive log events to measure token changes and augmented each *edit-instrument* event with tokens (i.e., edit-list, edit-function). Sometimes, participants added or deleted whole lines of code. We separated these kinds of changes from token changes, and compared consecutive code snapshots to determine these line change counts.

For trial and error, we counted the number of edits between plays (i.e., code executions) based on Dong et al.'s definition of test-based tinkering - *"Test-based tinkering is a type of behavior that involves executing the part of the code the student is editing. Students who exhibit test-based tinkering tend to test their scripts frequently within a few edits... students generally start with a script not working as intended, usually for an unknown reason. The student would make minor changes in the script followed by immediate testing in an attempt to fix the anomaly."* Thus, participants with lower number of edits between plays were viewed as exhibiting more test-based tinkering.

For RQ3, which compares tinkering behavior during daily tasks with behavior during the final competition capstone project, we compared the measures from RQ2 for the day 5 and day 8 activities against the same measures for the final competition capstone project. We performed the comparison only for participants who edited code in the daily activities.

5 FINDINGS

We present our findings for n=22 and n=16 middle schoolers in after-class music coding activities of day 5 and 8 of the online summer music coding camp, towards better understanding how we

can assess the development of CT concepts and practices during coding with music.

RQ1: What CT concepts are evident in participants' coding products? What percentage of participants met the requirements and recommendations of daily activities?

Table 3 presents our results for CT concepts embedded in final code products. The last two columns indicate the number (and percentage) of participants who met each requirement and followed each recommendation for each day's music coding activity. Requirements are labeled R, while recommendations are labeled C. A note in music is analogous to a variable in programming (e.g., C = 60), while a chord is implemented in TunePad as a list (e.g., F_maj = [F, A, C]). To illustrate, Figure 2 shows an example music code with chords and looping. Not in shown in the table is the fact that only one participant's code had a remaining syntax error.

```
1  D_maj7 = [ D, fSharp, A, cSharp ]
2  B_min9 = [ B, D, fSharp2, A ]
3  chords = [ B_min9, B_min9, D_maj7 ]
4  for chord in chords:
5      playNote(chord, beats=4)
```

Figure 2: Example music code with chords and loop

Overall, in addition to what is shown on Table 3, 9 of 38 participants (23.7%) met all requirements and followed all recommendations on both days (2/22 on day 5 (9.1%) and 7/16 on day 8 (43.8%)); 5 of 38 participants (13.2%) followed all recommendations but did not meet all requirements; none of the participants met all requirements but not followed all recommendations. Across the two days' activities, the majority of the participants showed success in defining (even nested) chords and using given chords. Approximately half of the participants used loops when they were recommended on day 8 and a few of them played functions with recommended chords. Only 10-50% ensured there were the required 4 beats in each measure. It is not surprising that participants did not define new or change notes as the initial code had predefined notes, and adding or changing notes was not part of the requirements or recommendations for these two days.

RQ2: What CT practices, in the form of tinkering behaviors, were exhibited in participants' process logs during music coding in daily activities?

Figure 3 shows the number of code changes (both token-based and added/deleted lines of code) per participant for day 5, day 8, and the final capstone project. We show the results separately for the token-based changes, which are related to the CT concepts that were covered by each day's after-class activity. Since tasks are different for each day, we are illustrating the results, but we are not comparing across days.

Overall, on average participants made between 2 and 20 code changes across both after-class activities. We believe this indicates a fairly broad range of construction-based tinkering among participants given the nature of the after-class activities, which required limited code. Participants exhibited more exploratory tinkering with functions and parameter tokens than lists and loops. We believe this is related to the requirements and recommendations which required and recommended fewer lists and loops than functions and parameters. Specifically for parameters, which are used in music coding primarily for beat length, we observed that less than 50% of the participants met the required 4 beats per measure, but they exhibited higher levels of exploratory behavior than other concepts. This may be due to creatively exploring different music flavors.

Table 3: Participants following requirements (R) and recommendations (C) in final music code product

Concept	Require/Recommend	Num of Participants			
			Day 5 (n = 22)		Day 8 (n = 16)
Note (Variable)	Define new notes		0		1 (6.3%)
	Change notes		1 (4.5%)		0
Chord (List)	Define required chords	R	15 (68.2%)		
	Define nested chords				10 (62.5%)
	Use given chords			R	15 (93.8%)
	Play recommend chords	C	7 (31.8%)		
Function	4 beats in each measure	R	2 (9.1%)	R	8 (50%)
	1-5 functions called		7 (31.8%)		
	6-10 functions called		4 (18.2%)		
	11-15 functions called		4 (18.2%)		
	15+ functions called		7 (31.8%)		
	0 function changed		14 (63.6%)		
	1-2 functions changed		7 (31.8%)		
	3+ functions changed		1 (4.5%)		
Loop	Use loop			C	9 (56.3%)

Figure 4 presents our test-based tinkering behavior findings for day 5 and 8 and the final capstone project in terms of number of edits between two melody plays. There was a fairly small spread between participants. The majority of participants made 3 to 5 edit events between two plays across both days. The low number of edits between plays suggests that participants were following test-based tinkering behavior during their music coding.

When we split the data by gender, we saw little difference between boys and girls in their tinkering behaviors. In addition to gender, our pre-camp survey also asked participants about their prior music experience and interest in computing. For both representative days, more than 60% of participants had prior music experience and approximately half of the participants had a prior interest in computing. We did not observe any significant differences in tinkering behaviors between different groups. The findings suggest that in this context, gender, prior music experience, and interest in computing have no correlation with tinkering behaviors.

RQ3: How did participants' tinkering behaviors during daily tasks compare with their tinkering behaviors during the final competition capstone task?

Figures 3 and 4 include the findings on tinkering behaviors during the final capstone project competition. Note that the x-axis in Figure 3 is a considerably different scale than the daily after-class activities, primarily due to the capstone project being larger in scope engaging participants considerably longer in making code changes and exploring music coding. However, the tinkering behaviors among the tokens and lines of code changes show similar

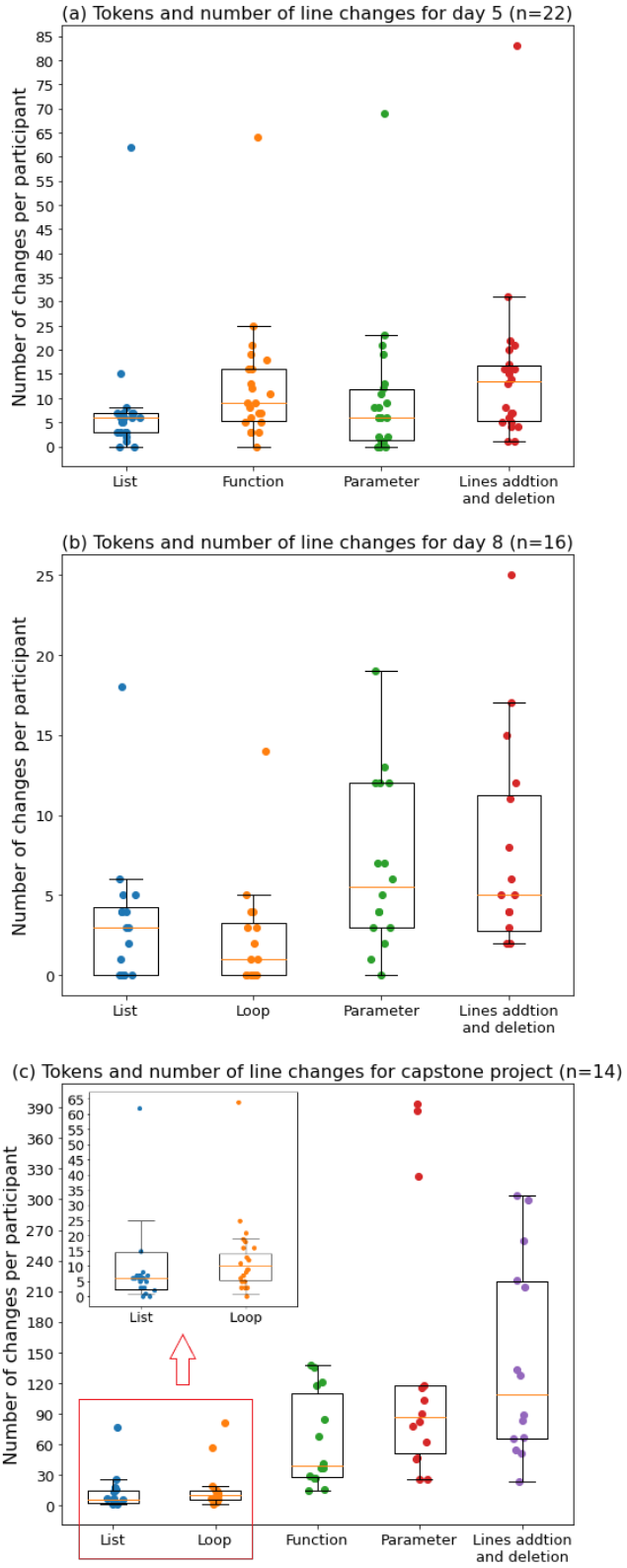

Figure 3: Construction-based, exploratory tinkering during after-class activities and capstone project

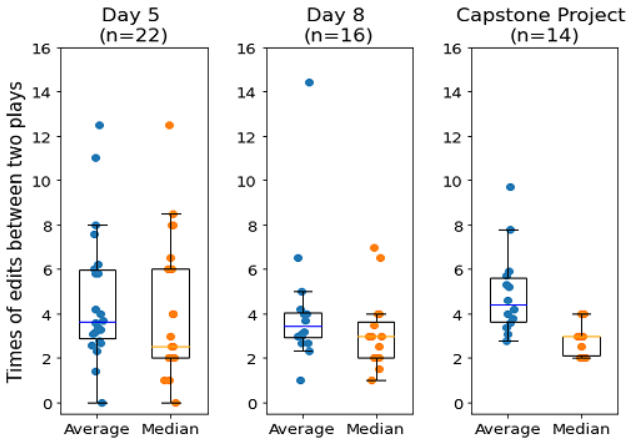

Figure 4: Test-based tinkering during after-class activities and capstone project

patterns. For example, similar to daily after-class activities, lists and loops are the two less frequently tinkered tokens. For other measurements, the range of the results (x-axis) is significantly larger than daily tasks. Figure 4 shows that in terms of test-based tinkering, the average number of edits between plays has a narrower range of variability among participants, but still in the same range of 2 to 6 edits between plays per participant in general.

6 SUMMARY AND CONCLUSIONS

To summarize, we expected more participants to meet the requirements and recommendations during the after-class activities. The overall lower than expected percentages may be due to the online nature of the camp, where students did not have in-person support for help and engagement, or the creative nature of music. We observed a broad range of construction-based tinkering among participants, which did not appear to be related to gender, prior music experience or prior interests in computing. Since a major goal of our work is to broaden participation in computing, this finding is encouraging. Nonetheless, we plan to investigate potential reasons for this variation. In contrast, overall, all participants demonstrated some test-based tinkering with few edits between plays.

One threat to validity is the small number of participants who actually edited code during the after-class activities. We plan to increase that number in future in-person offerings of the summer camp to gather more data. Moreover, we did not collect any qualitative data focusing on students' thinking processes while coding, which may be important for identifying CT practices. This limitation could be mitigated using think-aloud methods, as students frequently have difficulties recalling their thoughts after finishing a task [6, 10].

To our knowledge, this case study is one of the first to use process logs in music coding to explore CT practices in an integrated approach to CT learning. Future work includes collecting a greater volume of process log data that may help uncover additional patterns in the development of CT practices while coding with music.

ACKNOWLEDGMENTS

This paper is based upon work supported by the National Science Foundation under grants 2048793 and 2048792.

REFERENCES

[1] Sam Aaron. 2016. Sonic Pi–performance in education, technology and art. *International Journal of Performance Arts and Digital Media* 12, 2 (2016), 171–178.

[2] Yasemin Allsop. 2019. Assessing computational thinking process using a multiple evaluation approach. *International journal of child-computer interaction* 19 (2019), 30–55.

[3] Adriano Baratè, Andrea Formica, Luca A Ludovico, and Dario Malchiodi. 2017. Fostering computational thinking in secondary school through music-an educational experience based on google blockly. In *International Conference on Computer Supported Education*, Vol. 2. SCITEPRESS, 117–124.

[4] Valerie Barr and Chris Stephenson. 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 (2011), 48–54.

[5] Judith Bell and Tim Bell. 2018. Integrating computational thinking with a music education context. *Informatics in Education* 17, 2 (2018), 151–166.

[6] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, Vol. 1. 25.

[7] Pam Burnard, Zsolt Lavicza, Carrie Anne Philbin, et al. 2016. Strictly coding: Connecting mathematics and music through digital making. *Proceedings of Bridges 2016: Mathematics, Music, Art, Architecture, Education, Culture* (2016), 345–350.

[8] Yihuan Dong, Samiha Marwan, Veronica Catete, Thomas Price, and Tiffany Barnes. 2019. Defining tinkering behavior in open-ended block-based programming assignments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 1204–1210.

[9] Shelly Engelman, Brian Magerko, Tom McKlin, Morgan Miller, Doug Edwards, and Jason Freeman. 2017. Creativity in authentic STEAM education with EarSketch. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 183–188.

[10] John H Flavell. 1979. Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American psychologist* 34, 10 (1979), 906.

[11] Jason Freeman, Brian Magerko, Doug Edwards, Tom Mcklin, Taneisha Lee, and Roxanne Moore. 2019. EarSketch: engaging broad populations in computing through music. *Commun. ACM* 62, 9 (2019), 78–85.

[12] Gena R. Greher and Jesse M. Heines. 2014. *Computational Thinking in Sound: Teaching the Art and Science of Music and Technology*. Oxford University Press, Inc., USA.

[13] Mark Guzdial. 2003. A Media Computation Course for Non-Majors. *SIGCSE Bull.* 35, 3 (jun 2003), 104–108. https://doi.org/10.1145/961290.961542

[14] Michael Horn, Amartya Banerjee, Melanie West, Nichole Pinkard, Amy Pratt, Jason Freeman, Brian Magerko, and Tom McKlin. 2020. TunePad: Engaging learners at the intersection of music and code. (2020).

[15] Rachel Horst, Kedrick James, Yuya Takeda, and William Rowluck. 2020. From play to creative extrapolation: Fostering emergent computational thinking in the makerspace. *Journal of Strategic Innovation and Sustainability* 15, 5 (2020), 40–54.

[16] Ulas Ilic, Halil İbrahim Haseski, and Ufuk Tugtekin. 2018. Publication Trends over 10 Years of Computational Thinking Research. *Contemporary Educational Technology* 9, 2 (2018), 131–153.

[17] Fatemeh Jamshidi and Daniela Marghitu. 2019. Using music to foster engagement in introductory computing courses. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 1278–1278.

[18] Christian Köppe. 2020. Program a Hit–Using Music as Motivator for Introducing Programming Concepts. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 266–272.

[19] Samantha Krieger, Meghan Allen, and Catherine Rawn. 2015. Are females disinclined to tinker in computer science?. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 102–107.

[20] James Lockwood and Aidan Mooney. 2018. Computational Thinking in Secondary Education: Where Does It Fit? A Systematic Literary Review. *International Journal of Computer Science Education in Schools* 2, 1 (2018), n1.

[21] Douglas Lusa Krug, Edtwuan Bowman, Taylor Barnett, Lori Pollock, and David Shepherd. 2021. *Code Beats: A Virtual Camp for Middle Schoolers Coding Hip Hop*. Association for Computing Machinery, New York, NY, USA, 397–403. https://doi.org/10.1145/3408877.3432424

[22] Brian Magerko, Jason Freeman, Tom Mcklin, Mike Reilly, Elise Livingston, Scott Mccoid, and Andrea Crews-Brown. 2016. Earsketch: A steam-based approach for underrepresented populations in high school computer science education. *ACM Transactions on Computing Education (TOCE)* 16, 4 (2016), 1–25.

[23] Tom McKlin, Dana Wanzer, Taneisha Lee, Brian Magerko, Doug Edwards, Sabrina Grossman, and Jason Freeman. 2019. Implementing EarSketch: Connecting classroom implementation to student outcomes. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 634–640.

[24] Christopher Petrie. 2021. Interdisciplinary computational thinking with music and programming: a case study on algorithmic music composition with Sonic Pi. *Computer Science Education* (2021), 1–23.

[25] Thomas W Price, David Hovemeyer, Kelly Rivers, Ge Gao, Austin Cory Bart, Ayaan M Kazerouni, Brett A Becker, Andrew Petersen, Luke Gusukuma, Stephen H Edwards, et al. 2020. Progsnap2: A flexible format for programming process data. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 356–362.

[26] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (nov 2009), 60–67. https://doi.org/10.1145/1592761.1592779

[27] Daniele Traversaro, Giovanna Guerrini, and Giorgio Delzanno. 2020. Sonic Pi for TBL Teaching Units in an Introductory Programming Course. In *Adjunct Publication of the 28th ACM Conference on User Modeling, Adaptation and Personalization*. 143–150.

[28] Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.

[29] LeChen Zhang and Jalal Nouri. 2019. A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education* 141 (2019), 103607.