Behaviour & Information Technology

ISSN: 0144-929X (Print) 1362-3001 (Online) Journal homepage: www.tandfonline.com/journals/tbit20

Computational musicking: music + coding as a hybrid practice

Cameron L. Roberts & Michael S. Horn

To cite this article: Cameron L. Roberts & Michael S. Horn (2025) Computational musicking: music + coding as a hybrid practice, Behaviour & Information Technology, 44:5, 993-1013, DOI: 10.1080/0144929X.2024.2402533

To link to this article: https://doi.org/10.1080/0144929X.2024.2402533

Published online: 13 Sep 2024.

BEHAVIOUR & INFORMATION TECHNOLOGY
2025, VOL. 44, NO. 5, 993–1013
https://doi.org/10.1080/0144929X.2024.2402533

Taylor & Francis
Taylor & Francis Group

Computational musicking: music + coding as a hybrid practice

Cameron L. Roberts and Michael S. Horn

Computer Science and Learning Sciences, Northwestern University, Evanston, IL, USA

ABSTRACT

While there is a growing body of research that explores the integration of music and coding in learning environments, much of this work has either emphasised the technical aspects of computer language design or music as a motivational context within which to learn computer science concepts. In this paper, we report on a study in which five undergraduate students with experience in both music and coding completed two creative musical tasks: one using conventional instruments and tools and one using Python code in an online music + coding environment. Inspired by the work of Christopher Small (1998. *Musicking: The Meanings of Performing and Listening*. University Press of New England), we describe music + coding as a set of interlocking processes which we call *computational musicking* and explore how practices from both domains are reimagined in this new hybrid context. We introduce semiotic theories of translation and transcription to make sense of the computational musicking process and describe strategies that participants devised in their creative process.

ARTICLE HISTORY
Received 9 September 2023
Accepted 4 August 2024

KEYWORDS
Computational literacy; music + coding; STEAM education; music education

1. Introduction

Over the centuries, technological developments have profoundly impacted the musical aesthetics and practices of the Western musical tradition. Prior to the advent of written notation, the primary mechanism of musical transmission was oral tradition. Around the tenth century, neumatic notation introduced the use of markings to denote melodic contour – yet notably not exact pitch or rhythm (Strayer 2013). This written notation was a new form of technology that allowed for the development of a shared repertoire of song that could be transmitted across both distance and time. Further developments such as the familiar five-line musical staff allowed for greater fidelity notations that captured exact pitch and rhythm.

During the nineteenth century, improvements in instruments – for example, the development of the valve horn and the Boehm fingering system for flute – allowed composers to access new musical colours in the form of increased chromaticism (Tresch and Dolan 2013), which defined the music of the Romantic period. These innovations also afforded higher levels of musical virtuosity, which allowed artists to reach newfound levels of celebrity (e.g. 'Lisztomania'). New industrial and manufacturing processes that were developed during the twentieth century allowed for mass production of instruments. This in part led to the so-called 'saxophone craze' in which over a million saxophones were sold within a decade (Vermazen 2004).

But perhaps the most disruptive technology of all has been audio recording. The availability of recording technology starting in the late nineteenth century has had a profound impact on both popular music and art music. Recording allowed musical performances to be traded as a commodity and to even become mundane. In the domain of art music, recordings have allowed for the creation of canonical interpretations. They also enabled composers such as Pierre Schaeffer and John Cage to create so-called 'tape pieces' – musical works based entirely on existing recorded musical material. Today, the digital age has drastically changed how we consume music by making a seemingly infinite catalog of music readily available through streaming. Artists and researchers have long sought to harness the creative potential that exists at the intersection of music and computing (Lazzarini 2013; Wang 2017). In the days of mainframe computers, this exploration was restricted to large institutions with projects such as MUSIC-N at Bell Labs and ILLIAC at University of Illinois (Freeman 2022). With the invention of personal computers, advances in computational technology and

CONTACT Cameron L. Roberts cameronroberts2020@u.northwestern.edu Northwestern University, 2120 Campus Drive, Evanston, IL, 60208-2610, USA

programming language design have helped support thriving communities interested in the affordances of computer languages for expressing musical ideas and supporting creative expression. Examples include explorations of live coding (Aaron and Blackwell 2013), functional programming paradigms for representing musical structure (Aaron and Blackwell 2013), tools for modular synthesis (Wang 2017), and data-flow languages (Puckette 1996). Other research has focused on educational questions: *Can the intersection of music and code help students learn about music-making? Or computer programming? Can this intersection change students' relationships with both domains in terms of identity and self-expression?*

While there is a growing body of research exploring these questions, the field is in need of theoretical foundations that help conceptualise music and coding as bidirectional and mutually influencing. In this paper, we report on a study in which five undergraduate students with experience in both music and coding completed two creative musical tasks – one using traditional musical tools, instruments, and representations and one using the Python programming language and an online learning platform called TunePad (Horn, West, and Roberts 2022). Using a constructivist grounded theory approach (Charmaz 2006), we observe the critical role of *translation* (Eco 2008) between representational systems in the music + coding process and the potential for computation to serve as a restructuration of musical knowledge.

2. Related work

Of the many technical domains in which coding interacts, music is a fascinating case. Many of the domains that researchers have investigated rely on formal representational systems. Examples include physics (Sherin 2001), chemistry (Levy and Wilensky 2009), and the life sciences (Wilensky and Reisman 2006). Due to the technical nature of these domains, researchers are able to ask if code can supplement, augment, or even replace conventional representational systems, and, in the process, restructure learners' cognitive engagement with the concepts themselves (diSessa 2018; Sherin 2001; Wilensky and Papert 2010). While music involves formal and technical representational systems, it is also a sociocultural phenomenon that cannot be separated from existing encultured knowledge. Music-making, or *musicking* (Small 1998), is a pervasive human activity. We are surrounded by music from birth in countless different forms. Through this process of enculturation, individuals acquire 'basic musical competencies through everyday exposure to music' (Hannon and Trainor 2007). This manifests itself as an increased affinity and intuition for music from one's own culture through culture-specific cognitive representations. This contrasts with formal musical training, which gives individuals explicit power over this musical intuition by introducing new categories, abstractions, and processes. Research has also shown that individuals with musical training have improved cognitive abilities related to sound – such as pitch discrimination and tonal pattern recognition – which correlate with levels of expertise (Barrett et al. 2013). In other words, formal music instruction directly changes how we process and encode sound information. Therefore, in introducing code as a representation of music, this new computational literacy must come into contact with both encultured and formal aspects of an individual's musical literacy.

The use of music and code to enhance learning has a long history. Work from Jeanne Bamberger laid a conceptual foundation. Bamberger and colleagues developed MusicLOGO, which built upon the LOGO programming language by adding new instructions to play musical notes. Through work with MusicLOGO, Bamberger found that creating music procedurally shaped learners' perception of music (Bamberger 1979). Building on their work with MusicLOGO, Bamberger and colleagues developed a block-based environment called Impromptu with a lower barrier to entry for novice users. Impromptu allowed users to manipulate larger structural units of music such as phrases, which Bamberger calls 'units of perception' (Bamberger 1996). She speculates that the basic entity by which we understand music is not the note, but rather groups of notes.

Although work from Bamberger investigated how coding musical ideas shaped individuals' musical development, this has largely not been reflected in more recent music + coding work. Much of the recent work in music + coding falls under the umbrella of STEAM education – that is, science, technology, engineering, art(s), and mathematics. The STEAM movement has brought STEM learning into contact with artistic traditions and practices. Although these efforts ostensibly seek to combine arts and STEM education as equal partners, much of the work has overtly prioritised improving outcomes in STEM, particularly in populations that have been historically underrepresented in computing careers (Belbase et al. 2022; Perignat and Katz-Buonincontro 2019). In the realm of computing education, much of this arts + coding work has focused on cultivating computational thinking (Wing 2006; 2008; 2011). While definitions of computational thinking (CT) are nebulous (Kafai and Proctor 2022), many frame it cognitively as using foundational concepts and cognitive processes of computer science – such as abstraction,

iterative problem solving, and conditional logic – to solve both computational as well as non-computational problems (Grover and Pea 2013). Wing (2011) highlights the general applicability of CT as a skillset beyond computing careers, although this claim is disputed in the literature (Denning 2017).

Several general-purpose block-based programming environments such as Scratch (Resnick et al. 2009) and Microsoft MakeCode have included music-making capabilities. A notable example of this work in Scratch is from Greher and Heines (2014), who developed *Sound Thinking*, an interdisciplinary undergraduate course which explored connections between music and computational thinking. This programme sought to 'teach music and computing *together*, rather than one in service to the other' (Heines et al. 2011, 25). While these platforms have allowed millions of users to access high quality introductory coding education, they are not necessarily optimised for musical expression. Payne and Ruthmann (2019) offered three critiques of Scratch's music creation functionality. These critiques could be applied to similar general-purpose environments.

1. The numerical representation of pitch (MIDI) does not map easily to notes and rhythms, and these numerical representations may obscure melodic and rhythmic relationships.
2. Programming a song is tedious and requires music theoretical knowledge, which creates barriers to creative expression.
3. Because environments like Scratch are often designed around creating visual animations, the timing model can be unintuitive to users and compromise musical synchronisation.

This is to say perhaps that general-purpose programming languages like Scratch have a low barrier to *programming* but not to programming *music*. More recent domain-specific projects seek to address these concerns by creating platforms not only with a lower barrier to beginners, but also with features explicitly optimised towards music creation. Jython Music is a platform built on the Python programming language. It allows note-based composition as well as integration with external controllers such as smartphones and MIDI devices (Manaris, Stevens, and Brown 2016). Sonic Pi is a free platform built on the Ruby language that enables music creation and real-time performance with live coding (Aaron and Blackwell 2013; Aaron, Blackwell, and Burnard 2016). Findings of empirical studies with Sonic Pi have found increased confidence and engagement in students in both formal and informal contexts (Burnard, Lavicza, and Philbin 2016; Lusa Krug et al. 2021; Traversaro, Guerrini, and Delzanno 2020) as well as improved learning outcomes of programming concepts (Sinclair 2014). A study from Petrie (2022a) suggests that Sonic Pi can support both CT and music learning objectives, although the primary focus was assessing CT skills and practices in the context of music coding.

EarSketch and TunePad both support full-featured, browser-based programming environments modelled on digital audio workstations (DAWs) such as GarageBand, Pro Tools, and Ableton. EarSketch (Freeman et al. 2019; Magerko et al. 2016) introduces Python or JavaScript programming through sample-based composition with an expansive library of professional quality samples. This seeks to lower barriers to music creation by reducing the amount of music theory necessary in the music creation process. Quantitative studies with EarSketch have demonstrated both improved attitudes towards computing as well as an increased intention to persist in computing (Freeman et al. 2019; Magerko et al. 2016; Wanzer et al. 2019).

TunePad was developed as a companion to EarSketch and allows for both note-based and sample-based music creation with the Python programming language (Horn, Banerjee, and Brucker 2022; Horn, West, and Roberts 2022). TunePad uses a computational notebook paradigm modelled on systems like Jupyter and supports real-time collaborative editing between learners. Like Jython, TunePad supports integration with external MIDI controllers and allows for users to create custom event handlers for these captured MIDI events (Brucker, West, and Horn 2023). Studies with TunePad have shown increased engagement and improved attitudes towards computing (Gorson et al. 2017; Horn et al. 2020; Lusa Krug et al. 2023). Other work has included investigations of users' coding practices and usage of CT concepts (Zhang et al. 2022) as well as work evaluating the effectiveness of a system for giving automated feedback (Lusa Krug et al. 2023).

Two studies comparing TunePad and EarSketch found that the platforms have different strengths and weaknesses that educators should consider when choosing a platform, although the specific recommendations of the studies contradict each other. Petrie (2023) suggests that TunePad has 'a lower floor and ceiling' for users, while McCall et al.'s (2022) artifact-centered analysis found that TunePad users utilised higher levels of musical and computational complexity but that the barrier to entry (i.e. 'floor') might be higher. More comparative work is needed to parse out specific strengths and weaknesses specific platforms may have, and the degree to which this is generalisable.

Much of the prior work at the intersection of music and coding has either emphasised the technical aspects

of computer language design (e.g. Aaron and Blackwell 2013; Manaris, Stevens, and Brown 2016); or it has thought about music as a motivational context within which to learn computer science concepts and potentially broaden participation in computing fields (e.g. Lusa Krug et al. 2023; Magerko et al. 2016; Petrie 2022b). What music learning or creativity looks like in the hybrid domain has largely not been explored since Jeanne Bamberger's work with MusicLOGO and Impromptu from over 20 years ago. Studies that have concerned music learning have either conceived of a unidirectional relationship of computing acting on music (Petrie 2022a) or have looked for connections between the two separate domains (Bell and Bell 2018; Greher and Heines 2014). This is to say, music outcomes have not been systematically evaluated in the music + coding literature, and music has primarily served as a culturally-relevant material for computing education.

What's missing in the research on music + coding is work that considers the implications of music and computation as literacies that come into collision with one another. We use the term collision as a metaphor to conjure images of two massive objects slamming into one another and becoming permanently entangled. The result can be messy and confused. There are remnants of both original objects, but they have been deformed and transformed – revealing, from an artistic perspective, new creative potentials. Music and coding are both massive objects comprising technical complexity, weighty history, and cultural richness. How do these two domains act on one another, and what implications does this have for designing learning environments that utilise both? If conventional formal music instruction alters how learners encode and process sound, how might code? In our view, when literacies collide, transformative spaces can be opened in which both domains are reimagined. Combining coding and arts education – as music + coding systems such as TunePad, EarSketch, and Sonic Pi seek to do – can alter a learner's experience with both domains. On one hand, we are given new tools for understanding the algorithmic, mathematical, and structural aspects of music. But, extending our metaphor of collision, what is less often considered is that music might also change how we think about code.

In discussing 'coding music', 'music-coding', 'music + coding', or any of the other varied terms used in the space, we conflate the two meanings of the word *music*: as a noun and as a verb. Small's idea of *musicking* allows us to separate the activity of making and experiencing music from music as a mere product or commodity (Small 1998). We propose the term 'computational musicking' as a way to avoid this conflation and to emphasise the concept as an activity.

2.1. Literacies

2.1.1. Musical literacies

There have been many attempts to define musical literacy, both in connection with traditional language literacy (Hall and Robinson 2012; Hansen, Bernstorf, and Stuber 2007; Philpott 2015) and domain-specific conceptions (Broomhead 2021; Weidner 2018). Many conventional definitions of musical literacy, often foregrounded in music classrooms, focus on decoding musical notation as the primary competency (Mills and McPherson 2015). This view of musical literacy reflects a Western bias that often neglects many other forms of musical activity.

In our view, musical literacy should necessarily encompass the skills needed to support a diverse set of musical practices, taking into account the various cognitive processes that shape our interaction with sound as well as addressing the sociocultural context that a literacy implies. We conceive of musical literacy as the skills and competencies needed to interact and negotiate with a musical text in a culturally defined, substantive manner. These interactions may include conventional musical activities such as performing, composing, improvising, analyzing, and listening. This is also inclusive of participatory ways of interacting with music such as remixing, covering, satirising, or sampling (Tobias 2013). A musical literacy is necessarily specific to a musical culture rather than universal, although there may be overlap of competencies.

Rather than privileging symbolic forms, the ability to think abstractly in sound, or *audiation*, is the core competency of our concept of musical literacy. Audiation is the process of hearing and comprehending sound in one's mind when the sound is not present. This is separate from the concept of aural perception, which occurs when sound *is* present (Gordon 2012). Audiation may be engaged for a range of musical activities: imagining the resulting sound while reading sheet music, improvising, or writing music. Those who are skilled readers of musical notation also show a link between the notation and sound. That is, they can audiate or subvocalize the imagined sound of a piece without any intermediary representation. In contrast, more novice readers may link notation not with a sound result, but with a kinesthetic means of sound production – that is, for example, they may associate a note on the staff with a key on a piano or a specific fingering on the flute (Mills and McPherson 2015). Other supporting competencies include (but are not limited to) working

with theoretical and notational systems as well as instrumental proficiency and collaboration.

2.1.2. Computational literacies

We also think of coding as a form of literacy. Our perspective is derived from diSessa's notion of *computational literacy* (diSessa 2018; Li et al. 2020) a *technical literacy* with cognitive, social, and cultural dimensions. On a cognitive level, calling computation a *literacy* implies fluency with technical representation systems. This includes not only a deep familiarity with programming language syntax and semantics, but also a functional ability to make use of an array of associated tools. For example, programmers may make use of documentation, a variety of online resources, and tools such as IDEs (integrated development environments). Importantly, in diSessa's formulation, this technical fluency also reflects back on us, shaping how we think about the world, how we solve problems, and how we engage in creative tasks. As a person becomes more computationally literate, they may be able to think more abstractly in terms of tools, design patterns, or approaches to solving a problem without worrying about language-specific syntax or formalisms.

On a social level, computational literacy implies an aspect of communication and sharing of ideas that is facilitated through technical inscriptions. The shared understanding of technical inscriptions supports communities of people engaged in a variety of endeavours. Examples might include a team of software engineers communicating about a project; developers sharing problems and solutions through online communities such as Stack Overflow; or scientists publishing computational notebooks (e.g. Jupyter) along with more traditional manuscripts to share digital datasets and the computations performed on those datasets. As such communities grow, they also develop distinct cultures and subcultures.

Finally, diSessa argues that literacies should have society-wide implications that transcend any one domain or discipline. For this last point, diSessa might then expect distinct *genres* of computational literacy to emerge when applied to specialised tasks. In our case, this would imply that we could expect to see new genres of computational literacy begin to form around music creation, production, and performance. Building on the work of diSessa and others, Kafai and Proctor (2022) call for a shift to computational literacies to account for sociocultural as well as critical perspectives on computing. Adopting a literacies perspective allows for multiple conceptions both across and within different communities. This heterogeneity (Rosebery et al. 2010; Sengupta, Dickes, and Voss Farris 2021) is core to our conception of both computational and musical literacies.

Our notions of musical literacy and computational literacy overlap in several ways. For instance, both require working with abstractions. In music, this may involve recognising chord progressions or melodic motifs, while in computation, it may involve recognising common programming structures such as loops or functions. Both may involve specialised symbolic languages (such as sheet music or Python code) as well as iterative problem-solving routines (i.e. practicing and debugging).

2.2. Creative thinking

Perhaps the most important way in which these musical literacies are deployed is through creative expression. Research on creativity has identified four distinct conceptions: creativity as a personality trait, as a product, as a place, and as a structured process of working (Kozbelt, Beghetto, and Runco 2010; Sawyer 2018). Early research of creativity emphasised creativity as a personality trait and sought to create tools to assess the aptitudes and cognitive abilities that contribute to an individual's creative potential (e.g. Guilford 1956; Torrance 1974). Conceptions that focus on the creative product often emphasise the novelty of the end result of this process, while definitions focusing on place largely focus on the environmental factors and social context (Amabile 2011; Csikszentmihalyi 2015). The process definition of creativity emphasises the sequential stages through which creative ideas emerge and are developed; this definition highlights creativity as a dynamic and iterative process that unfolds through a series of interconnected stages, ultimately leading to an artifact (e.g. Halverson, 2013; Sawyer 2021; Wallas 1926).

Sawyer (2018, 2021) found the process model of creativity widely utilised in his study of pedagogical beliefs within professional schools of art and design. Sawyer's cultural model of arts education describes a pedagogical approach in which learners are guided through an iterative, exploratory, and improvisational process. Similarly, Sheridan et al. (2014) described the creative process within three makerspaces as an iterative process of working with tools, ideas, and material. Halverson (2013) described the process of digital art making in terms of a *representational trajectory*, wherein individuals must consider the relationship between their intended idea and the representational affordances of the medium, ultimately moving towards more refined representations.

Specifically addressing the ways in which children think creatively in music, Peter Webster developed a

widely utilised process model of creative thinking in music: creative thinking in music can be viewed as a structured and recursive process of moving from *divergent* and *convergent* thinking towards the creation of a musical product (Hickey and Webster 2001; Webster 1990). Divergent thinking refers to a mode of thought which can produce many different possibilities and is not directed towards a single goal; convergent thinking, on the other hand, is a mode which evaluates possibilities and converges upon a solution. Creative thinking is influenced by a range of constraints, including representational, cultural (i.e. theoretical systems), social, and those inherent to the parameters of the specific task (Burnard and Younker 2002). In terms of the constraints of representation, this might include the affordances of instruments, notation, or technological interfaces (Bamberger 2006). Beyond these external constraints, enabling skills and conditions affect the creative process on a personal level. These can include aptitudes, musical literacy, motivation, and aesthetic taste (Webster 2002).

Webster identifies four stages of creative thinking through which individuals can move in a non-linear and recursive manner, adapted from a model devised by Graham Wallas (Wallas 1926):

- **Preparation**: formulate and/or consider the problem and gather materials and ideas.
- **Incubation**: generate with a range of musical ideas, considering multiple possibilities but not yet settling on a single idea.
- **Illumination**: evaluate the multiple possibilities and settle on a single idea.
- **Verification**: verify the musical decisions through playback or some other means.

As we will show in our Findings section below, computational literacy also became entangled in this creative process when participants wrote Python code to develop musical ideas.

2.3. Translation

To mediate between the two representational systems associated with these forms of literacy, we chose the analogy of *translation*, building on the work of Umberto Eco (Eco 2008). This choice is due in part to the ambiguities that are inherent in making a translation. To use language as an example, this is to say that meaning is fundamentally altered in making a translation from one language to another, both due to cultural differences and the abstractions that are available in each language. Through the process of translation, meaning – which may or may not accurately reflect the meaning of the original text – is transmuted and reconstructed in the new language.

Eco's model of translation expands the work of Jakobson (1959) to encompass translation both between and within languages and other sign systems. *Intrasemiotic* translation (or intralinguistic in the case of language) is a reformulation within a single sign system. In language, this may manifest as rewording of a thought. *Intersemiotic* translation (interlinguistic in language) encompasses translation between sign systems. This may manifest as a change of media, such as adapting a novel to the stage. Both forms of translation can be contrasted with *transcription*, which Eco describes as being able to be carried out by 'automatic substitution', such as moving from a natural language to morse code (Eco 2008).

In the context of this paper, we are taking the set of musical and computational signs to be our two principal systems of interest – that is the set of signs and relations that convey musical and computational meanings. Because we are considering the set of signs, an intrasemiotic translation in this context could be considered moving from one computational representation to another – for instance, refactoring code or moving from one coding language to another. Similarly, an intrasemiotic translation in music may consist of moving from one form of musical notation to another, such as from tablature to sheet music. We consider intersemiotic translation to be the transfer of knowledge from one system to another. In computation, this could manifest as the implementation of a computer programme (i.e. moving from a list of requirements to code) (Loddo, Addis, and Lorini 2022). In this example, the programmer moves from the textual representation, which describes the programme's functionality, to a coded representation. A related example might involve implementing a graphical user interface. The programmer may begin with a visual representation (in the form of a wireframe or paper prototype) which they then represent as code. In both of these examples, implementation of the computer programme requires the programmer to interpret the initial representation of the programme and to make decisions about the structure and syntax of the coded representation. In music, this form of translation could consist of adapting a novel to the stage as an opera. This adaptation may require the translator to reconfigure the narrative structure of the novel to fit operatic conventions and to develop musical motifs to represent characters and themes.

In each of these examples of translation, the translator navigates a representational process in which they

must consider the semantics of what is represented as well as the affordances of both representational domains. This process necessarily requires them to draw on their own interpretive skills and creative thinking, as well as their prior experience in both domains.

3. Methods

The following subsections provide an overview of our participants, data collection, and analysis methods.

3.1. Participants

All five of our participants were undergraduate students participating in a two-week training programme to prepare coaches for teaching summer programmes with TunePad. Each coach had some amount of experience with both coding and music. Each had at least one STEM major as well as some level of formal Computer Science background. Four participants were involved in informal musical activities (e.g. a cappella or songwriting), and one participant was studying viola performance as a second major. Two of the participants had been TunePad coaches the previous summer and had prior experience with TunePad.

Participant	Musical background	STEM background
Jan	Singer-songwriter (guitar, piano)	2nd year CS/math major
Lewis	A cappella	2nd year CS/cognitive science major
Ian	Music Performance major (viola)	5th year CS major
Morgan	High school band (trombone)	5th year CS major (B.S./M.S.)
Sal	High school band (saxophone) Amateur guitarist	4th year CS major

3.2. Data collection

The five coaches participated in two sessions, each around 45 min long and on separate days. For both sessions, participants were given approximately 20 min to compose a short original melody and then reflected on their compositional process through a semi-structured interview. For the first session, participants were instructed to bring any non-coding tools of their choice to aid their composition. For the second, participants were instructed to code an entirely new melody using TunePad. Participants were not cut off after twenty minutes and could continue working on their composition until they felt it was complete. For the first session, participants worked for an average of 24 min; for the second session, they worked for an average of approximately 28 min.

Participants were not informed of the specific musical task they were to complete ahead of each session to avoid any potential preparation. Every participant provided signed informed consent as well as verbal consent to be video recorded. While composing, participants were instructed to think aloud about their musical process as well as to vocalise musical ideas by humming or singing. Think-aloud protocols (Ericsson and Simon 2002) are a widely utilised tool for gaining insight into participants' thought processes and problem-solving strategies. In think-aloud interviews, the participant is asked to complete a task while vocalising their thoughts by 'skimming the surface' of their consciousness.

In the reflection portion of the interview, participants were asked to describe their musical process in greater detail. They were asked questions such as: *Why did you choose your specific musical tool? How did you structure your musical composition? What strategies did you use to write your musical ideas as code? How do your coding skills transfer into music + coding? How do your music + coding strategies differ from your regular coding strategies?*

Interviews were audio and video recorded and then transcribed for analysis.

3.3. Research questions

The purpose of this study was to investigate how computation affected the musical creative process and how music affected the coding process. The following questions guided the analysis:

1. How did participants engage in both the musical and computational aspects of the task?
2. How did processes and artifacts between the trials differ?
3. What are the limitations and affordances of code as a form of musical representation?
4. How did the nature of the musical task affect the code that participants created?

3.4. Analysis and positionality

To make sense of our data, we applied a constructivist grounded theory approach (Charmaz 2006), considering both interviews as well as the musical artifacts created. Transcripts were inductively coded sentence-by-sentence; this initial open coding phase produced 130 unique codes: for example, *using formal music*

theory knowledge (N = 61), *scaffolding code (N = 10)*, *debugging coded composition (N = 25),* and *experimenting with musical ideas (N = 67)*. After initial coding, transcript data was edited for clarity and conciseness, removing filler words and redundancy. Then initial codes were grouped into larger emergent themes which were refined and abstracted through constant comparison. These categories included *musical representations* (e.g. notational or instrumental), *musical literacy* (both formal and informal), *computational literacy*, *compositional process*, and *knowledge transfer*. Individual data were segmented and aggregated into the major thematic areas and categories were once again refined. Through the successive phases of analysis, the role of *translation* between musical and computational representations emerged, as well as strategies participants utilised in making these translations.

A key component of this approach is acknowledging the researcher's own subjectivity in building a theory. Both authors are white, male-identifying scholars with training in computer science who have contributed substantially to the TunePad software platform and curriculum. For this study, the first author led data collection and analysis activities, and both authors contributed to interpreting the findings. The first author is a member of multiple communities of music makers, both in Western classical and American folk musical traditions. He holds an advanced degree in music performance, which has provided him with training predominantly within a Western classical theoretical framework. He is an active composer and performer specialising in experimental music. In his compositions, he is particularly drawn to alternative musical representations, including text, graphic, and hybrid scores. This involvement in diverse musical communities informs his scholarship. The second author is a tenured professor of computer science and learning sciences. He created the TunePad platform and was conducting the coach training workshop.

3.5. Tunepad

In this study we used TunePad, a free, online music + code environment (Horn et al. 2020). This research could also have been conducted using similar music + code environments that make use of text-based programming languages. TunePad projects take the form of specialised computational notebooks called playbooks that consist of an arbitrary number of cells containing Python code, multimedia content, and text. Code in TunePad can represent individual musical elements such as notes, rests, chords, or percussion sounds. The basic primitive of TunePad is the *playNote* function, which takes parameters for MIDI note and duration in beats. The *playNote* function moves an abstraction called the playhead, which dictates where notes are placed in time. Users can also manipulate this playhead directly in code with functions like *moveTo*, *rewind*, and *fastForward*. The core TunePad library also provides access to mixing and mastering effects such as panning and filtering. In addition to this standard library, users also have access to additional modules such as the *constants* modules, which defines variables for note names (e.g. Bb2, C4, A5) and rhythmic values (e.g. WHOLE_NOTE, HALF_NOTE), and the *chords* modules, which allows users to build and play diatonic chords. Users also have access to a user submitted library of code and sounds that they can import and use in their own compositions.

4. Findings

For the first task, participants created a variety of musical compositions. Jan (pseudonym), a singer-songwriter and musician, created a country-western inspired verse using GarageBand, MIDI keyboard, and her voice. Ian, a classically trained violist, composed a simple classical melody using his viola. Lewis, an *a cappella* singer and arranger, created an indie-pop inspired song with two contrasting sections using a MIDI keyboard and voice. Sal, an amateur guitarist and saxophonist, wrote a full verse-chorus song using guitar and voice. Morgan, who participated in high school band and college *a cappella,* has no formal training in music theory. Morgan used online reference material as well as a MIDI keyboard, notation software, and whistling to compose a short melodic fragment.

For the second task, participants used TunePad and Python code to create an entirely new composition. Jan started by creating a drum and bass accompaniment and then composed an eight-measure pop piano melody over that accompaniment. Ian wrote three separate motifs inspired by electronic dance music (EDM) but struggled to line up the different cells rhythmically. Like Ian, Lewis created short motifs rather than developing a longer melody; he wrote four different motifs which were each two measures in duration. Sal wrote a two-measure piano motif outlining two chords and a drum beat and then composed a four-measure piano melody over this texture; like Ian, Sal also struggled to rhythmically align the different cells. Morgan was the fastest to complete his melody; he first developed a short, eight-measure chord progression and then composed a piano melody on top of this.

Across both tasks, we began to see the role that *translation* – both conscious and automatic – played in

negotiating representations of both music and code. Particularly in creating their coded compositions, participants drew on a range of musical representations in developing and notating their musical ideas. Their musical ideas were initially developed separately from code and then iteratively translated into code. The translation to a coded representation of music served as both a barrier but also as an avenue for new creative possibilities, and participants utilised novel strategies to make this translational process easier. In addition to employing new strategies to adapt to the specific demands of music + coding, working in a hybrid environment also altered conventional computational practices, such as debugging and refactoring.

In the subsequent sections, we will describe this process of translation, as well as the strategies participants utilised; we will then contextualise this process of translation in our model of *computational musicking*. Before we offer our detailed analysis, we will begin by presenting two in-depth vignettes of participants completing the computational musicking task.

4.1. Vignette: Jan

'I'm going to start off with a simple drum beat', Jan says as she begins defining variables for the drum sounds she will use in TunePad. She quickly writes code for a 'four-on-the-floor' drum beat, one of the most common in popular music, starting with the bass drum and snare. Then she uses the built-in *rewind* function – which allows for concurrent musical lines – to add hi-hats on each eighth note inside of a for-loop.

She creates a *Bass* cell and noodles around on the virtual instrument. 'Now I'm going to come up with a bassline, but I'm just trying to figure out what key I want. I'm kind of thinking about it, and humming out loud', she says while vocalising a simple stepwise melodic line. 'I'm gonna just do like a … bass note to give me a concept of the chords that I'm writing the melody over to give it some sort of structure'. Instead of coding the notes right away, Jan leaves comments with the names of the chords she envisions. After creating this scaffold for herself, she codes the bassline with a slightly syncopated rhythm. She listens to the audio output of her code and makes a face. Her rhythms do not quite line up with the musical measures, but she makes a change to the beats values and listens back again to confirm that this did the trick.

> "You know what? I'm going to do an AABA structure. I'll define this [bassline] as a function." She indents her bassline to encapsulate it into a new function and then defines a stub function where she will create the B section. "I'm going to have this play a slightly different chord progression than my A section."

Once she's happy with her bassline, Jan begins working on her melody by creating a *Keyboard* cell. '[The keyboard] is a little bit easier for me. I don't really play the bass, but I do play the piano'. She plays the drums and bass and starts to hum a melody over it. She clicks around on the keyboard to find her starting note.

Jan quickly translates her melodic idea by coding the opening measure. Continuing on, she uses a combination of singing and playing back on the keyboard to figure out notes' MIDI values. Once she has aurally established her key, she mentally calculates MIDI notes based on her intervals. Singing and tapping out beats, she calculates her beats values. With the first phrase coded, she listens back, ensuring that the notes and rhythms she coded match what was in her head.

> I don't know if it's worth it at this point, but I could create variables like 'eighth = 0.5' so that I can say I want an eighth note. That helps me visualize a little better. It's an easier connection if I'm thinking about sheet music. But I guess I could also switch to the sheet music view.

She sings her melody and compares it to the sheet music visualisation in order to verify its correctness. It only takes her a moment to realise that she has again made an error with the beats values. She makes a few changes, playing it back after each one to see if it's fixed.

'For the next one, I'm going to do something a little different', she says as she copies and pastes her code. It only takes her a minute to make a few tweaks, singing and using the piano to figure out the notes. She repeats this process for her final two phrases as well.

4.2. Vignette: Morgan

'Let's start playing chords that I know will sound good. Then I'll try to add some fun ones once I'm feeling more confident'. Morgan quickly writes code for a chord progression using the built-in *playChord* function using roman numeral notation. He extends each chord to play for four beats and listens to what he has so far. He isn't happy, so he starts adding more chords. 'I know a V/V chord sometimes sounds good!' He experiments with a V/V but quickly deletes it.

'I think I want this [chord] to be voiced higher so that there are more upper notes in there as well'. To figure out notes are being played, he logs the output of the *buildChord* (which operates similarly to *playChord* but returns a list of MIDI values in a chord) to the console. He begins to write a list comprehension expression to add an octave to each note in the chord so that it

plays in both octaves but hesitates for a moment. 'Actually, I wonder if I can pass the *octave* parameter a list rather than just a number'. He gives this a try and is happy to see that it works as he hypothesised. He listens back to his chord progression to decide whether it is good enough for him to move on.

> "I realized that I inverted my V chord the wrong way and made it higher. I'll lower the chord before it [as well] so that way it sounds like [the progression] is moving down." He switches the track he is working in from the piano roll visualization to sheet music and continues to tinker with the voicing for a few minutes, trying to get a relatively smooth bassline with his chords. "Okay, I think this will do."

Now that he has a solid harmonic foundation, Morgan creates a new *Keyboard* cell to contain the code for his melody.

> I don't really know the notes each chord is built of, so I'm going to print each of them to the console with *buildChord*. This will help me have a sense of what notes I can use in the melody.

He copies and pastes the code from his harmony and changes the function to output the lists of notes to the console. Then, he pastes this output as a comment. 'This might not even be that helpful because I'm going to use the *constants* [module]'. He imports the *tunepad.constants* module, which defines variables which map note names to MIDI values.

Morgan plays his harmony and hums a simple melody over the chords. He starts coding the first measure and whistles his idea a few times before finding his starting note on the keyboard. The code he writes doesn't exactly match the idea he whistled, but he is still happy with how it sounds and continues. 'I don't know why I'm hearing an A in the second measure. Let's see if it sounds good'. He plays it back and smiles. 'I guess I was just hearing the seventh'.

Knowing he is running short on time, he hurriedly writes code for the next two measures of music. 'I'll just quickly get something down and then go back and tweak it'. He largely abandons the musical idea he had originally come up with and falls back onto using the chord tones in the harmony and his ear to guide the remainder of the melody. His music theory knowledge allows him to add scalar passing tones and ornamentation on this outline. He speeds through the last few measures without much difficulty.

4.3. Translation and the creative process

In the vignette above, Jan narrowed her melodic options by choosing a bassline and drum beat and then allowed her intuition to guide her in writing the melody. She was guided by her ear more than by music theory, although her knowledge of music theory was a powerful tool in translating her musical ideas into code. In crafting her musical ideas, she used non-coding tools much in the same way she did in her first task.

Morgan, on the other hand, used a blend of technical terms from both music theory and computer science in his compositional process. He had the fluidity to switch between both systems with relative ease. Although he has a good grasp of musical concepts such as chord theory and melodic writing, he admitted that he lacks the aural skills to make the sound output of his code match the musical idea he has in his mind. Of the participants, he was the most fluid in TunePad but lacked some of the informal musical intuition that comes from experience. His knowledge of the TunePad platform allowed him to augment his theoretical knowledge and make music easier (e.g. the *chords* and *constants* modules).

Morgan's debugging and musical experimentation were highly linked. Most of his experimentation manifested as changing the specific parameters of functions between potential 'correct' or musically viable solutions (rather than converging on a correct solution). He chose between musical options he knew were theoretically correct and tweaked the parameters until he found something he was happy with.

4.3.1. Translation and encoding

In both the first and second task, we found that participants had to negotiate a variety of constraints, representations, and sensory-perceptual resources including:

- **Sound**: the aural feedback a participant receives
- **Theoretical knowledge**: both the intuitive and formalised means participants have of structuring musical knowledge.
- **Cultural knowledge**: the aesthetics and values enculturated both by specific genres/subcultures and of larger Western culture.
- **Instrumental**: the link between the sensorimotor network and perception.
- **Notational**: written systems for representing music, such as standard music notation and tablature.

Even in the non-coding task, participants had to *translate* between different musical texts and representations. This is to say, they were 'reading' or interpreting musical texts and negotiating musical meaning across related representational forms such as notation, instruments, and recorded media. We see this negotiation as

an *intrasemiotic translation*, or a reformulation within a single sign system – in this case, music.

This translation manifested in many different ways, depending on the representations in play. For instance, for some participants the physical instruments (e.g. guitar or viola) afforded for automaticity in which there is no conscious translation between thought and action. This automaticity is of course predicated on some degree of virtuosity, otherwise there must be an additional translational step. In the reflection after her second task, Jan remarked how this extra step could serve as a barrier to creative expression:

> I feel like I have so much control over my voice. When I was playing the saxophone, if I was trying to do a saxophone solo, I would have to think what I want to hear and how [to] translate that to my fingers and actually play it. If I really practiced my scales and got to that point, like if you're like an insane saxophone player, you kind of just think it and play. You don't do it in two steps; you just do that one step. That's how I feel a little bit with guitar and piano.

The music + coding process in the second task involved moving from a variety of musical forms – notational, instrumental, cultural, and others – to the language of computation through what we saw as a recursive process of refining towards higher-fidelity representations. As in the first task, translation was also present in how participants negotiated non-coding constraints and representations; in this instance, however, the participants demonstrated *intersemiotic* forms of translation: that is, movement from one sign system to another. We observed that in their compositional processes, none of the participants formulated musical ideas in code, but instead converged on their musical ideas through a combination of audiating and vocalising. While writing a drum beat, Ian spoke in terms showing the delineation between the musical and coding processes:

> In my head, I'm trying to think of what rhythm I want. It's probably going to be something like 'Boom. Clap. Boom, boom, clap.' It's pretty standard. Let's see if I can replicate that.

Jan spoke in similar terms to Ian:

> For my melody, I played through my bassline a few times and hummed over it – which is exactly what I do when I write music normally. I'll hum when I'm playing guitar or playing piano, and I see what I like. When I hear something that I like, I'm like, 'okay, I'll write this down.'

In this hybridised music + coding process, participants leveraged musical representations in the TunePad platform – including standard music notation, piano roll, and track instruments (keyboard, guitar, and drums). As we saw with both Jan and Morgan, participants took advantage of a wide range of musical representations. They used the track instruments both to sketch out musical ideas as well as to map musical notes to their MIDI values. They used the piano roll and sheet music visualisations to aid in verifying the correctness of their musical output and to help with debugging.

Participants then engaged in an active process in which semi-formed musical ideas were transformed from cognitive auditory structures (which may also leverage the appropriation of other forms of musical representations) into computer code. In other words, they enacted an informal process in which they transformed and encoded their musical idea into computer code.

4.3.2. Reinterpretation and evaluation

As part of their computational musicking process, the output of the code itself (i.e. the sound produced by TunePad) was evaluated and compared to the original musical idea of the participant. If necessary, the participant engaged in debugging and refining this code. This step was predicated on the participant being able to discern an incongruity or dissonance between their initial idea and the audio output of the code (related to musical literacy) as well as their ability to understand and debug the code they wrote (related to computational literacy). Although the software generated the musical output, in order to debug their musical idea, participants had to take the additional step of mapping the aural incongruity to its code representation. That is, this process required them to both read and understand the *code* as *music*. The translational process continued iteratively until the participant was able to find coherence between their mental representation of the sound and the musical artifact.

Computational and musical literacies work in tandem in a computational musicking setting, but the ability to make the translational step between systems can serve as a major pain point. Similar to the example of instrumental proficiency serving as a barrier to creative expression, coding fluidity has obvious implications for the success of a computational musicking task, as one participant remarked:

> [There's] a clear difference between the interns who had CS experience and [those] who didn't. The ones who had experience could make a lot cooler stuff, but I think that's just because we could do it faster and didn't have to be like, 'I want this thing to happen; how do I do it?' For me, it's 'I want something to happen. I kind of know how to do it because I know how coding works.'

In the example above, the participant reflected on how one's degree of computational knowledge can act as a constraint on musical expression; this can certainly work in reverse as well. Morgan, who was one of the most adept coders, lamented that his lack of music theory knowledge was his primary challenge in both tasks. His musical strategy involved a large degree of trial-and-error (divergent thinking), testing out many ideas without clear direction. This was time-consuming and caused frustration:

> Because all the aural skills that I had are gone, when I hear what would sound right in my head, there's no translation from that sound into a chord, or a knowledge of what comes next. So, there are things where I'm like, 'I wish that sounded like something else.' I would've either spent more time playing around with stuff, but in the time I had, playing around didn't work.

The process of translating to code also at times diverged from the intended outcome, generating new creative possibilities. The act of searching for coherence (and sometimes failing) often revealed new ideas, as Morgan expressed:

> Very [little] of what I wrote down is what I was hearing [in my head]. It's more like, 'oh, this sounds good anyway.' I didn't mean to mis-translate that, but I think in the end, I was able to make more efficient decisions where I was making more intelligent guesses.

4.4. Translational strategies

Beyond coding or musical competencies, the ability to make translational steps is dependent on building conceptual links across representation systems and domains. Participants took advantage of the abstractions and representational affordances of code to help with this process. Several participants used variables and constants to represent musical concepts such as note names and rhythmic values or used comments and line breaks to separate smaller musical units. Some participants also decomposed the problem of translation into subproblems. These strategies not only had the effect of aiding in creating the translation, but also made the code look 'more musical', as one participant remarked.

In our programming with K-12 students, understanding the mathematical and fractional nature of music rhythms has often been a pain point. For example, a quarter note in Western musical notation translates into 1 beat in TunePad Python code. An eighth note translates into 0.5 beats, and so on. The concept that beats are conventionally subdivided by powers of 2 (½, ¼, ⅛) is a difficult concept that is far from immediately obvious to beginners. Even an experienced music + coding participant struggled to make the link between music notation and beats without use of a translational aid. Speaking about this particular pain point, Ian says:

> You have to really focus on the math side of music in terms of the length of the notes and the articulation you want. […] In terms of just straight up note length, that's a lot easier to do as a musician. […] Coming from a music background, I have the intuition that a quarter of a beat is a 16th note, and a half a beat is an eighth note. But all that kind of gets thrown out the window.

One such strategy that other participants used to address this particular pain point was through use of the built-in *constants* library – which defines variables for notes and rhythmic values – or by defining their own variables (see Figure 1). One participant remarked that code written in this way 'looked like music' even though he understood that it was functionally the same. Another, that using variables in this way allowed them to leverage their knowledge of sheet music to better visualise the end product. This is not to mention the added bonus in terms of readability and explainability.

The organisational affordances that participants leveraged were not limited to variables. Comments allowed them to leave notes and reminders for things such as the notes that were part of the harmony. Participants also added line breaks and whitespace as an analogue to musical barlines by adding them in between measures. Morgan said,

> I like being able to separate what each measure looks like and write little notes to myself. It's nice and helpful. I kind of wanted this [in task #1] but writing out the notes of the chords would have been a nice tool.

The parameterised nature of coding allowed participants to decompose the problem of translation into two separate problems: translating pitches and translating rhythmic values. One participant described how they filled in placeholder values for rhythmic values to denote relative length before they calculated the exact value:

> Generally, I get the notes first and then deal with the rhythms after … Sometimes if I know that one note is shorter than the next one, I will just put like 0.5 instead. It's like I'm doing little annotations and then going back and editing.

4.5. Transforming coding practices

We observed that computational musicking also altered typical computational practices, particularly related to debugging and refactoring. Debugging is a software engineering practice in which people work through

Code without variables	Equivalent code with variables for notes	With variables for notes and duration
playNote(48, beats=1)	from tunepad.constants import *	from tunepad.constants import *
playNote(48, beats=1)		QUARTER = 1.0
playNote(55, beats=1)	playNote(C3, beats=1)	HALF = 2.0
playNote(55, beats=1)	playNote(C3, beats=1)	
playNote(57, beats=1)	playNote(G3, beats=1)	playNote(C3, beats = QUARTER)
playNote(57, beats=1)	playNote(G3, beats=1)	playNote(C3, beats = QUARTER)
playNote(55, beats=2)	playNote(A3, beats=1)	playNote(G3, beats = QUARTER)
	playNote(A3, beats=1)	playNote(G3, beats = QUARTER)
	playNote(G3, beats=2)	playNote(A3, beats = QUARTER)
		playNote(A3, beats = QUARTER)
		playNote(G3, beats = HALF)

Figure 1. Use of variables as a translational aid.

code to diagnose, locate, and solve problems with code. Refactoring is a process of iteratively improving code without altering its underlying functionality. Refactoring might include introducing variables, functions, or other abstractions that result in code that is less redundant, easier to understand, more efficient, and less prone to errors.

4.5.1. Debugging and multiple hearings

In the process of translation, participants used multiple forms of debugging. On one level, they practiced conventional debugging: both syntactic and semantic (Vessey 1986). Syntax errors in programming occur when the structure of the code violates the language's rules, preventing proper interpretation or compilation, while semantic errors are caused by mistakes in the logic of a programme that may not produce syntax errors but lead to undesired or incorrect behaviour. In debugging these conventional programming bugs, participants were able to use the Python interpreter and their computational knowledge. In the evaluation of the correctness of their code (i.e. the output matching their musical ideas), they were required to employ an entirely different form of debugging through musical aural skills. They had to find errors in the programme through sound, first identifying the incongruity in sound and then mapping that onto code. One participant remarked on the difference between this auditory debugging and conventional debugging:

> The way that I'm testing is by listening; it's more of an interactive way. How I would normally debug is [through] test code or maybe a print statement or [with] an output. It's kind of like this is the output except instead of visually seeing an output, I'm hearing it and processing myself. [Instead of] the computer telling me what's wrong, I'm telling me what's wrong.

In the debugging process, listeners must manage and remedy multiple incongruent hearings of a musical idea in order to find coherence as well as contend with the ephemeral nature of musical memory. Bamberger characterises musical development in terms of this sort of disequilibrium among representations (Bamberger 2006):

> Musical development is enhanced by continuously evolving interactions among multiple organizing constraints along with the disequilibrium and sensitivity to growing complexity that these entanglements entrain […] Rather than being a uni-directional process, musical development is a spiraling, endlessly recursive process in which organizing constraints such as those above are concurrently present creating an essential, generative tension as they play a transformational dance with one another. (page 4)

This process has the capacity to engage both musical and computational literacies and also to reinforce both. As we saw with our participants, the dissonance between the mental idea and its computational form can highlight the ambiguities in a listener's mental representation and results in a spiraling process in which both are refined.

4.5.2. Refactoring as restructuration

Key to our concept of translation is the fact that coding is itself a creative process in which there are multiple

'correct' solutions; that is, there are multiple ways to formulate code which have the same musical output. Unlike canonical musical notations, the difference between these code representations is not syntactical but structural – although we have also described strategies employed on the level of syntax (i.e. comments, variables). There are heuristics for evaluating and choosing between different viable solutions – such as readability, modularity, and conciseness – and we saw participants actively employing these heuristics in the refinement of their code. There are certainly translations which obscure the musical structure in the exact same way as musical notation (see Figure 2 below). Sequential representations – essentially a list of notes and rhythmic values – are arguably indistinguishable in form from standard musical notation (but are nonetheless an important stage towards thinking programmatically about music). These are arguably more akin to a *transcription* rather than a translation (Eco 2008).

Although the time constraints of the task discouraged some participants from refining their code, more advanced coders were able to encapsulate elements of musical structure using coding abstractions such as loops and functions. Morgan describes this as a matter of convenience:

> [A]s soon as I find a pattern – which is very, very common in popular music – my code starts to look much more modular because it's easy to break things up and make my life easier.

Another participant echoed this preference for convenience and conveyed that they were evaluating their code based on metrics of readability and conciseness:

> I defined two little functions – my A and B functions – so I don't have to write these over and over again. Even though I'll only use the B function once – I wouldn't need to create a function because I'm not using it again – it's nice to have it more organized. I think it's pretty important when you're writing music in TunePad being able to use variables, comments, functions because it really does clean it up and make it tidier and easier.

As this shows, the process does not end when participants reach a 'correct' answer – and in fact, as we have seen, it may stop before then – it ends when they reach coherence between the musical idea and its representation.

We view computer code as a *restructuration* – or, a shift in the representational infrastructure – of musical knowledge (Wilensky and Papert 2010). According to Wilensky and Papert, a change in structuration has the capacity to transform a learner's relationship with content as well as to improve its learnability. Using the example of the shift from Roman to Arabic numerals, they illustrate how this shift in representation facilitated the gradual development of widespread arithmetic skill. Within music, this computational and mathematical musical formulation has the capacity to reveal the underlying formal and mathematical structure of music – what Bamberger might call 'units of perception' (Bamberger 1996). The process of translation required participants to reflect on this musical structure on multiple levels: both micro-level rhythmic timing as well as larger phrase units. As Bamberger and diSessa (2003) found with music and mathematics, working with this formulation caused insights about the underlying structure, which is obscured by other forms of notation. This is to say that the translational process serves as a form of musical analysis.

We do not wish to suggest, however, that computational representations of music should supplant other musical representations. As we have observed, work in a computational musicking environment was very much rooted in participants' prior musical knowledge and existing musical intuition. Rather, we wish to acknowledge that computing has unique affordances that impact *what* is learned in a hybrid learning environment. As Papert wrote in *Mindstorms*, computing serves to 'recast powerful ideas in computational form' (Papert 1993, 183). In this way, we see computation as a means for grasping the 'powerful idea' of musical form.

4.6. Creative thinking in computational musicking

Even though the process of generating musical ideas seemed similar in the non-coding and coding sessions, coding added additional complicating steps on top of the musical creative process. In the non-coding task, participants generated their musical ideas through an iterative and improvisational process. Participants primarily worked with a single musical tool or representation in generating their musical ideas. Working with their chosen musical tool, participants experimented with many different musical ideas and gradually built their song. These ideas were ephemeral; the melodies largely existed only in the heads of the participants and were not performed the same way twice. Participants spontaneously introduced expressive variations in how they performed their melodies. In this way, their musical artifacts defied notation, which serves to create a canonical codified musical text, as these improvisatory expressive gestures that were introduced in performance are not able to be captured by fixed notation.

Although the DAW-like coding environment of TunePad did not afford for the same degree of

TunePad code for *Hot Crossed Buns*

```
playNote(52, beats = 1)
playNote(50, beats = 1)
playNote(48, beats = 2)
playNote(52, beats = 1)
playNote(50, beats = 1)
playNote(48, beats = 2)
playNote(48, beats = 0.5)
playNote(48, beats = 0.5)
playNote(48, beats = 0.5)
playNote(48, beats = 0.5)
playNote(50, beats = 0.5)
playNote(50, beats = 0.5)
playNote(50, beats = 0.5)
playNote(50, beats = 0.5)
playNote(52, beats = 1)
playNote(50, beats = 1)
playNote(48, beats = 2)
```

Refactored code to encapsulate repeating sections

```
def phrase_a:
    playNote(52, beats = 1)
    playNote(50, beats = 1)
    playNote(48, beats = 2)

def phrase_b:
    playNote(48, beats = 0.5)
    playNote(48, beats = 0.5)
    playNote(48, beats = 0.5)
    playNote(48, beats = 0.5)
    playNote(50, beats = 0.5)
    playNote(50, beats = 0.5)
    playNote(50, beats = 0.5)
    playNote(50, beats = 0.5)

phrase_a()
phrase_a()
phrase_b()
phrase_a()
```

Figure 2. Use of encapsulation to illuminate musical structure.

immediacy and spontaneity as the tangible and embodied musical tools that participants used in the first task, notating their musical ideas in code required them to interrogate and decompose their musical ideas. Participants had to engage in a translational process to make their musical ideas explicit in code. In translating their musical ideas, participants drew on a greater number of different musical representations to create their own formal representation of their musical ideas.

Unlike standard western notation – which maps musical ideas to a discrete set of symbols that represent mathematical relationships – participants had to encode these mathematical representations directly. Creating the formal and mathematical representations of their musical ideas required them to interrogate and refine their understandings of those ideas (Cox 1999; Halverson, 2013). That is, they debugged both the code and their own mental representation or 'hearing' of the idea.

For our participants, translation served as a parallel and sometimes intertwined process with musical creative thinking – thus the computational musicking process was more of a variation of conventional musical creative thinking than something entirely different. We have thus far identified the following concurrent and intersecting processes (Figure 3):

1. **Musical creative thinking**: an iterative and recursive creative process to generate a musical idea
2. **Encoding**: mapping of musical sign systems to computer code
3. **Sound debugging**: identifying and fixing musical errors and incongruities between code and original musical idea
4. **Conventional debugging**: solving syntactical, logical, and semantic problems with code
5. **Refinement**: applying coding heuristics (such as readability, modularity, abstractions, comments, and conciseness) to improve code

In this formulation, encoding, sound debugging, refinement, and even conventional debugging all involve forms of intersemiotic translation.

For our participants, musical creative thinking served as the first step, but after this participants moved fluidly between these different processes. The musical ideas that the participants came up with became problems to solve with code – acting as a sort of analog to programme requirements in a conventional coding setting. Once

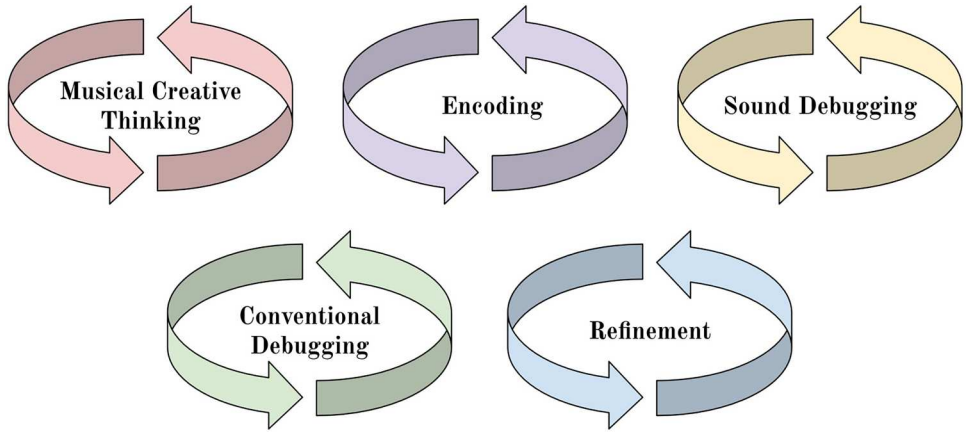

Figure 3. Constituent processes of computational musicking.

they had solidified their musical ideas, they began to deconstruct and encode them as constituent notes and rhythmic values. They made use of musical tools (such as whistling, strumming, playing a keyboard, or looking at sheet music), theoretical knowledge, and novel translational techniques. When engaging in sound debugging, participants read their code as music to fix incongruities between the computer-generated sound output and their own musical ideas. When refining their code, participants made use of common software engineering heuristics and techniques, such as introducing variable names in place of hard-coded values, reducing redundant code with loops or functions, and adding comments. Even though these heuristics and techniques are common in software engineering practice, our participants employed them in the service of revealing an underlying musical structure that may not have been previously conscious or visible in the other representation systems that they used. This evolving process was directed towards the refinement of the original musical idea, but these translational processes could also diverge into new ideas by happy accident.

There is no set path for how an individual might move through these processes, and there is no requirement that they engage in every process. Not all of our participants engaged in refinement, and it is easy to imagine a scenario where a more advanced coder would not have to undertake conventional debugging. Some individuals may move through in a more linear fashion, such as is illustrated in Figure 4. Others may take more convoluted paths which revisit past steps and diverge, such as in Figure 5.

5. Discussion and future work

In this paper, we have outlined the role of translation as a key practice of music + coding. In doing so, we propose that music + coding introduces shifts in both forms of literacy and begins to emerge as a hybrid of the two. We observed participants changing their practices to accommodate the specific considerations of the new hybrid domain. Changes operated on the levels of procedure, structure, and aesthetics of both code and music. That is, the *processes* of music-making and coding as well as the *artifacts* of music and code were altered. We introduce the term *computational musicking* to address the change and emphasise music + coding as a set of intertwined processes. On the level of aesthetics, participants sought to make their code 'look more musical' through imposing additional organisational constraints. These constraints replicated music-specific abstractions such as note names and barlines. On the level of procedure, participants leveraged language and platform affordances to aid in the translational process. On the level of structure, participants encapsulated elements of musical structure into loops and functions. Our findings suggest that prior music or coding training did not always transfer to the new music + coding context, even for participants experienced in *both* music and coding. Musical knowledge and familiarity with a particular musical representation are culturally specific and are often acquired through enculturation, which means that skills and knowledge that are specific to one context may not be directly transferable to another context. Success was rather stipulated on the cultivation of links between both domains.

In music + coding, this cultural specificity means that it is not enough simply to drop music into coding curricula or contexts. This involves recognising that music + coding is not an amalgamation of the two content areas, but rather a unique domain that requires specialised skills. In our music + coding task, musical creative thinking formed the foundation of knowledge generation, but this was highly interconnected with the

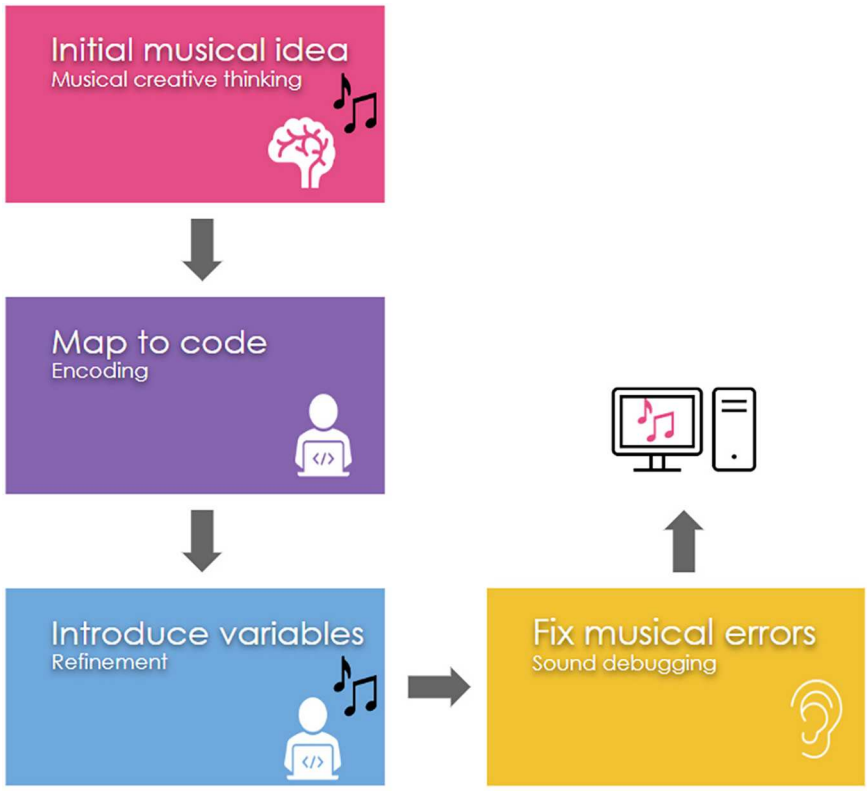

Figure 4. A linear computational musicking pathway.

coding process – with various forms of translation serving as mediation. The interplay between musical and computational knowledge allowed for new divergent creative possibilities, as we observed with Morgan. The refinement of both musical and computational representations that was achieved through the translational process suggests promising pedagogical implications for both domains. Similar to Halverson's (2013) *representational trajectory* in digital art making – which resulted in a progressive formalisation of

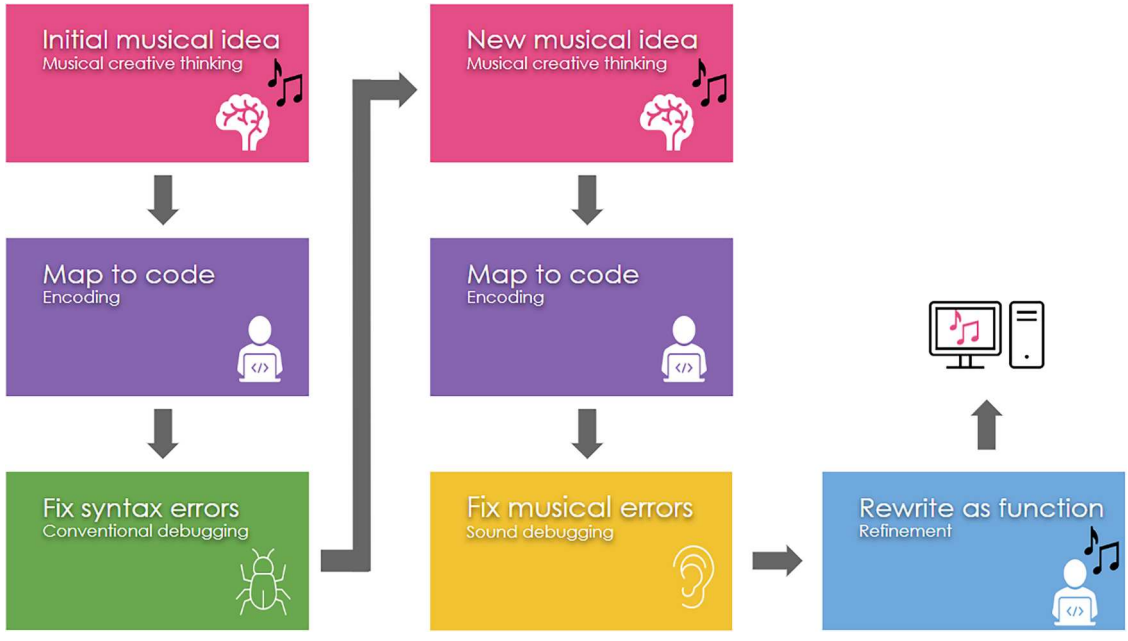

Figure 5. A more complex computational musicking pathway.

representations – we view the refinement of translations as a potential mechanism for learning in music + coding education. This, in turn, has the capacity to spur the development of both musical and computational literacies: musical literacy, through the illumination of these larger structural components; computational literacy, through the utilisation of principles of encapsulation, abstraction, and decomposition. More research is needed to understand the nature of learning in computational musicking across different time scales and contexts, as well as the role that translation plays. In particular, longitudinal work in naturalistic settings is necessary for a more complete picture of learning in the hybrid context as well as how it may differ from that of conventional music and coding instruction.

The implications of our findings suggest that curricula should focus on actively cultivating this translational link between musical knowledge and coding. Educators should aim to equip students with the necessary tools to navigate the complexities of musical representation in the coding context. This involves developing a deep understanding of musical concepts such as rhythm, melody, harmony, and timbre through coding and non-coding musical activities, as well as the ability to translate these concepts into code.

Instruction oriented primarily towards programming concepts will miss out on the full richness of arts + coding (STEAM) educational environments. In this paper, we have attempted to show this in the case of computational musicking through focusing on the representational forms at play. We have sought to get beyond a reified musical product through describing computational musicking as a set of interlocking processes. In our analysis, we described five processes that participants undertook. Only one process – conventional debugging – is principally rooted in computational knowledge. More advanced forms of computational thinking, namely encapsulation and abstraction, were triggered by *musical* problems. For example, using a list so that the participant could play multiple notes in the harmony; or, encapsulating musical structure into functions. For this reason, we postulate that any music + code environment that neglects the musical aspect is doing so at the expense of the educational potential of both domains.

A limitation of this study was sample size and participant demographics. We also note that the task used in this research design was artificially constrained and may not resemble how people use similar platforms to make music in real life. As such, we lack thick descriptions that could be gained from an ethnographic immersion within creative learning environments and communities. In future studies, we would like to expand our dataset to include participants from more diverse musical backgrounds and a wider set of ages and abilities as well as to compare multiple music + coding platforms. We plan to describe in more detail the different strategies that participants took in navigating their creative process with code as well as explore the implications of these findings on the design of our interfaces and curricula.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This research was supported by grants DRL-1612619, DRL-1451762, DRL-1837661, and DRL-2119701 from the National Science Foundation.

ORCID

Cameron L. Roberts http://orcid.org/0000-0003-1448-0829
Michael S. Horn http://orcid.org/0000-0002-4892-6801

References

Aaron, S., and A. F. Blackwell. 2013. "From Sonic Pi to Overtone: Creative Musical Experiences with Domain-Specific and Functional Languages." In *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling & Design*, 35–46. https://doi.org/10.1145/2505341.2505346.

Aaron, S., A. F. Blackwell, and P. Burnard. 2016. "The Development of Sonic Pi and its Use in Educational Partnerships: Co-creating Pedagogies for Learning Computer Programming." *Journal of Music, Technology & Education* 9 (1): 75–94. https://doi.org/10.1386/jmte.9.1.75_1.

Amabile, T. 2011. *Componential Theory of Creativity*. Boston, MA: Harvard Business School.

Bamberger, J. 1979. "Logo Music Projects: Experiments in Musical Perception and Design." *LOGO Memo* 52. http://hdl.handle.net/1721.1/5726.

Bamberger, J. 1996. "Turning Music Theory on its Ear do We Hear What We See; Do We See What We Say?" *International Journal of Computers for Mathematical Learning* 1 (1): 33–55. https://doi.org/10.1007/BF00191471.

Bamberger, J.. 2006. "What Develops in Musical Development? A View of Development as Learning." In *The Child as Musician: A handbook of musical development*. 1st edition, edited by Gary McPherson, 69–92. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780198530329.003.0004.

Bamberger, J., and A. diSessa. 2003. "Music as Embodied Mathematics: A Study of a Mutually Informing Affinity." *International Journal of Computers for Mathematical Learning* 8 (2): 123–160. https://doi.org/10.1023/B:IJCO.0000003872.84260.96.

Barrett, K. C., R. Ashley, D. L. Strait, and N. Kraus. 2013. "Art and Science: How Musical Training Shapes the Brain." *Frontiers in Psychology* 4: 1–13. https://doi.org/10.3389/fpsyg.2013.00713.

Belbase, S., B. R. Mainali, W. Kasemsukpipat, H. Tairab, M. Gochoo, and A. Jarrah. 2022. "At the Dawn of Science, Technology, Engineering, Arts, and Mathematics (STEAM) Education: Prospects, Priorities, Processes, and Problems." *International Journal of Mathematical Education in Science and Technology* 53 (11): 2919–2955. https://doi.org/10.1080/0020739X.2021.1922943.

Bell, J., and T. Bell. 2018. "Integrating Computational Thinking with a Music Education Context." *Informatics in Education* 17 (2): 151–166. https://doi.org/10.15388/infedu.2018.09.

Broomhead, P. 2021. "A New Definition of Music Literacy: What, Why, and How?" *Music Educators Journal* 107 (3): 15–21. https://doi.org/10.1177/0027432121991644.

Brucker, M., M. West, and M. Horn. 2023. "Digital Drum Circles." In *Relational CS Education Through Music Making. Proceedings of the 22nd Annual ACM Interaction Design and Children Conference*, 705–708. https://doi.org/10.1145/3585088.3594490.

Burnard, P., Z. Lavicza, and C. A. Philbin. 2016. "Strictly Coding: Connecting Mathematics and Music through Digital Making." *Proceedings of Bridges 2016: Mathematics, Music, Art, Architecture, Education, Culture* 345–350.

Burnard, P., and B. A. Younker. 2002. "Mapping Pathways: Fostering Creativity in Composition." *Music Education Research* 4 (2): 245–261. https://doi.org/10.1080/1461380022000011948.

Charmaz, K. 2006. *Constructing Grounded Theory*. London: Sage Publications.

Cox, R. 1999. "Representation Construction, Externalised Cognition and Individual Differences." *Learning and Instruction* 9 (4): 343–363. https://doi.org/10.1016/S0959-4752(98)00051-6.

Csikszentmihalyi, M. 2015. *The Systems Model of Creativity: The Collected Works of Mihaly Csikszentmihalyi*. Dordrecht: Springer.

Denning, P. J. 2017. "Remaining Trouble Spots with Computational Thinking." *Communications of the ACM* 60 (6): 33–39. https://doi.org/10.1145/2998438.

diSessa, A. A. 2018. "Computational Literacy and 'The Big Picture' Concerning Computers in Mathematics Education." *Mathematical Thinking and Learning* 20 (1): 3–31. https://doi.org/10.1080/10986065.2018.1403544.

Eco, U. 2008. *Experiences in Translation*. Translated by A. McEwen. University of Toronto Press. https://books.google.com/books?id=0dVYaP9VukIC.

Ericsson, K. A., and H. A. Simon. 2002. *Protocol Analysis: Verbal Reports as Data*. rev. ed., 4. pr. Cambridge: The MIT Press.

Freeman, J. 2022. "History of Music and Computing." In *Introduction to Digital Music with Python: Learning Music with Code*, edited by M. Horn, M. West, and C. Roberts, 194–206. New York City: Routledge.

Freeman, J., B. Magerko, D. Edwards, T. Mcklin, T. Lee, and R. Moore. 2019. "EarSketch: Engaging Broad Populations in Computing through Music." *Communications of the ACM* 62 (9): 78–85. https://doi.org/10.1145/3333613.

Gordon, E. 2012. *Learning Sequences in Music: A Contemporary Music Learning Theory*. GIA Publications. https://books.google.com/books?id=wPjRMgEACAAJ.

Gorson, J., N. Patel, E. Beheshti, B. Magerko, and M. Horn. 2017. "TunePad: Computational Thinking through Sound Composition." In *Proceedings of the 2017 Conference on Interaction Design and Children*, 484–489. https://doi.org/10.1145/3078072.3084313.

Greher, G. R., and J. M. Heines. 2014. *Computational Thinking in Sound: Teaching the Art and Science of Music and Technology*. Oxford: Oxford University Press.

Grover, S., and R. Pea. 2013. "Computational Thinking in K–12: A Review of the State of the Field." *Educational Researcher* 42 (1): 38–43. https://doi.org/10.3102/0013189X12463051.

Guilford, J. P. 1956. "The Structure of Intellect." *Psychological Bulletin* 53 (4): 267–293. https://doi.org/10.1037/h0040755.

Hall, S. N., and N. R. Robinson. 2012. "Music and Reading: Finding Connections from Within." *General Music Today* 26 (1): 11–18. https://doi.org/10.1177/1048371311432005.

Halverson, E. R. 2013. "Digital Art Making as a Representational Process." *The Journal of the Learning Sciences* 22 (1): 121–162. https://doi.org/10.1080/10508406.2011.639471.

Hannon, E. E., and L. J. Trainor. 2007. "Music Acquisition: Effects of Enculturation and Formal Training on Development." *Trends in Cognitive Sciences* 11 (11): 466–472. https://doi.org/10.1016/j.tics.2007.08.008.

Hansen, D., E. D. Bernstorf, and G. M. Stuber. 2007. *The Music and Literacy Connection*. Lanham, MD: Rowman & Littlefield Education.

Heines, J., G. Greher, S. Ruthmann, and B. Reilly. 2011. "Two Approaches to Interdisciplinary Computing + Music Courses." *Computer* 44 (12): 25–32. https://doi.org/10.1109/MC.2011.355.

Hickey, M., and P. Webster. 2001. "Creative Thinking in Music." *Music Educators Journal* 88 (1): 19–23. https://doi.org/10.2307/3399772.

Horn, M., A. Banerjee, and M. Brucker. 2022. "TunePad Playbooks: Designing Computational Notebooks for Creative Music Coding." In *CHI Conference on Human Factors in Computing Systems*, 1–12. https://doi.org/10.1145/3491102.3502021.

Horn, M., A. Banerjee, M. West, N. Pinkard, A. Pratt, J. Freeman, B. Magerko, and T. McKlin. 2020. *TunePad: Engaging Learners at the Intersection of Music and Code*.

Horn, M., M. West, and C. Roberts. 2022. *Introduction to Digital Music with Python: Learning Music with Code*. New York City: Routledge.

Jakobson, R. 1959. "On Linguistic Aspects of Translation." In *On Translation*, edited by R. A. Brower, 232–239. Cambridge: Harvard University Press. https://doi.org/10.4159/harvard.9780674731615.c18.

Kafai, Y. B., and C. Proctor. 2022. "A Revaluation of Computational Thinking in K–12 Education: Moving Toward Computational Literacies." *Educational Researcher* 51 (2): 146–151. https://doi.org/10.3102/0013189X211057904.

Kozbelt, A., R. A. Beghetto, and M. A. Runco. 2010. "Theories of Creativity." In *The Cambridge Handbook of Creativity*, edited by J. C. Kaufman and R. J. Sternberg, 20–47. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511763205.004.

Lazzarini, V. 2013. "The Development of Computer Music Programming Systems." *Journal of New Music Research*

42 (1): 97–110. https://doi.org/10.1080/09298215.2013.778890.

Levy, S. T., and U. Wilensky. 2009. "Students' Learning with the Connected Chemistry (CC1) Curriculum: Navigating the Complexities of the Particulate World." *Journal of Science Education and Technology* 18 (3): 243–254. https://doi.org/10.1007/s10956-009-9145-7.

Li, Y., A. H. Schoenfeld, A. A. diSessa, A. C. Graesser, L. C. Benson, L. D. English, and R. A. Duschl. 2020. "Computational Thinking is More about Thinking than Computing." *Journal for STEM Education Research* 3 (1): 1–18. https://doi.org/10.1007/s41979-020-00030-2.

Loddo, O. G., A. Addis, and G. Lorini. 2022. "Intersemiotic Translation of Contracts into Digital Environments." *Frontiers in Artificial Intelligence* 5:963692. https://doi.org/10.3389/frai.2022.963692.

Lusa Krug, D., E. Bowman, T. Barnett, L. Pollock, and D. Shepherd. 2021. "Code Beats: A Virtual Camp for Middle Schoolers Coding Hip Hop." *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 397–403. https://doi.org/10.1145/3408877.3432424.

Lusa Krug, D., C. Mouza, W. M. Jones, T. Barnett, and D. C. Shepherd. 2023. "Attracting Adults to Computer Programming via Hip Hop." In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education Vol. 1*, 528–534. https://doi.org/10.1145/3545945.3569800.

Magerko, B., J. Freeman, T. Mcklin, M. Reilly, E. Livingston, S. Mccoid, and A. Crews-Brown. 2016. "EarSketch: A STEAM-based Approach for Underrepresented Populations in High School Computer Science Education." *ACM Transactions on Computing Education* 16 (4): 1–25. https://doi.org/10.1145/2886418.

Manaris, B., B. Stevens, and A. R. Brown. 2016. "JythonMusic: An Environment for Teaching Algorithmic Music Composition, Dynamic Coding and Musical Performativity." *Journal of Music, Technology and Education* 9 (1): 33–56. https://doi.org/10.1386/jmte.9.1.33_1.

McCall, L., J. Freeman, T. McKlin, T. Lee, M. Horn, and B. Magerko. 2022. "Complementary Roles of Note-oriented and Mixing-oriented Software in Student Learning of Computer Science Plus Music." *Computer Music Journal* 46 (3): 48–66. https://doi.org/10.1162/comj_a_00651.

Mills, J., and G. E. McPherson. 2015. "Musical Literacy: Reading Traditional Clef Notation." In *The Child as Musician*, edited by G. E. McPherson, 177–191. Oxford: Oxford University Press. https://doi.org/10.1093/acprof:oso/9780198744443.003.0009.

Papert, S. 1993. *Mindstorms: Children, Computers, and Powerful Ideas*. 2nd ed. New York City: Basic Books.

Payne, W., and S. A. Ruthmann. 2019. "Music Making in Scratch: High Floors, Low Ceilings, and Narrow Walls." *Journal of Interactive Technology and Pedagogy* 15:2019.

Perignat, E., and J. Katz-Buonincontro. 2019. "STEAM in Practice and Research: An Integrative Literature Review." *Thinking Skills and Creativity* 31:31–43. https://doi.org/10.1016/j.tsc.2018.10.002.

Petrie, C. 2022a. "Interdisciplinary Computational Thinking with Music and Programming: A Case Study on Algorithmic Music Composition with Sonic Pi." *Computer Science Education* 32 (2): 260–282. https://doi.org/10.1080/08993408.2021.1935603.

Petrie, C. 2022b. "Programming Music with Sonic Pi Promotes Positive Attitudes for Beginners." *Computers & Education* 179:104409. https://doi.org/10.1016/j.compedu.2021.104409.

Petrie, C. 2023. "Design and use of Domain-specific Programming Platforms: Interdisciplinary Computational Thinking with EarSketch and TunePad." *Computer Science Education*, 1–34. https://doi.org/10.1080/08993408.2023.2240657.

Philpott, C. 2015. "Musical Literacy: Music as Language." In *The Child as Musician*, edited by G. E. McPherson, 192–207. Oxford: Oxford University Press. https://doi.org/10.1093/acprof:oso/9780198744443.003.0010.

Puckette, M. 1996. "Pure Data: Another Integrated Computer Music Environment." In *Proceedings of the Second Intercollege Computer Music Concerts*, 37–41.

Resnick, M., J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, et al. 2009. "Scratch: Programming for All." *Communications of the ACM* 52 (11): 60–67. https://doi.org/10.1145/1592761.1592779.

Rosebery, A. S., M. Ogonowski, M. DiSchino, and B. Warren. 2010. "'The Coat Traps All Your Body Heat': Heterogeneity as Fundamental to Learning." *Journal of the Learning Sciences* 19 (3): 322–357. https://doi.org/10.1080/10508406.2010.491752.

Sawyer, R. K. 2018. "Teaching and Learning How to Create in Schools of Art and Design." *Journal of the Learning Sciences* 27 (1): 137–181. https://doi.org/10.1080/10508406.2017.1381963.

Sawyer, R. K. 2021. "The Iterative and Improvisational Nature of the Creative Process." *Journal of Creativity* 31:100002. https://doi.org/10.1016/j.yjoc.2021.100002.

Sengupta, P., A. C. Dickes, and A. Voss Farris. 2021. *Voicing Code in STEM: A Dialogical Imagination*. 1st ed. Cambridge: The MIT Press.

Sheridan, K., E. R. Halverson, B. Litts, L. Brahms, L. Jacobs-Priebe, and T. Owens. 2014. "Learning in the Making: A Comparative Case Study of Three Makerspaces." *Harvard Educational Review* 84 (4): 505–531. https://doi.org/10.17763/haer.84.4.brr34733723j648u.

Sherin, B. L. 2001. "A Comparison of Programming Languages and Algebraic Notation as Expressive Languages for Physics." *International Journal of Computers for Mathematical Learning* 6 (1): 1–61. https://doi.org/10.1023/A:1011434026437.

Sinclair, A. 2014. "Educational Programming Languages: The Motivation to Learn with Sonic Pi." *Psychology of Programming Interest Group* 25: 1–14.

Small, C. 1998. *Musicking: The Meanings of Performing and Listening*. Hanover: Wesleyan University Press.

Strayer, H. R. 2013. "From Neumes to Notes: The Evolution of Music Notation." *Musical Offerings* 4 (1): 1–14. https://doi.org/10.15385/jmo.2013.4.1.1.

Tobias, E. S. 2013. "Toward Convergence: Adapting Music Education to Contemporary Society and Participatory Culture." *Music Educators Journal* 99 (4): 29–36. https://doi.org/10.1177/0027432113483318.

Torrance, E. P. 1974. *Torrance Tests of Creative Thinking: Verbal Tests, Forms A and B ; Figural Tests, Forms A and B; Norms-Technical Manual*. Princeton: Personnel Press.

Traversaro, D., G. Guerrini, and G. 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. https://doi.org/10.1145/3386392.3399317.

Tresch, J., and E. I. Dolan. 2013. "Toward a New Organology: Instruments of Music and Science." *Osiris* 28 (1): 278–298. https://doi.org/10.1086/671381.

Vermazen, B. 2004. *That Moaning Saxophone: The Six Brown Brothers and the Dawning of a Musical Craze: The Six Brown Brothers and the Dawning of a Musical Craze*. Oxford: Oxford University Press.

Vessey, I. 1986. "Expertise in Debugging Computer Programs: An Analysis of the Content of Verbal Protocols." *IEEE Transactions on Systems, Man, and Cybernetics* 16 (5): 621–637. https://doi.org/10.1109/TSMC.1986.289308.

Wallas, G. 1926. *The Art of Thought*. United Kingdom: Solis Press.

Wang, G. 2017. "A History of Programming and Music." In *The Cambridge Companion to Electronic Music*, edited by N. Collins and J. d'Escrivan, 2nd ed., 55–71. Cambridge: Cambridge University Press.

Wanzer, D., T. McKlin, D. Edwards, J. Freeman, and B. Magerko. 2019. "Assessing the Attitudes towards Computing Scale: A Survey Validation Study." In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 859–865. https://doi.org/10.1145/3287324.3287369.

Webster, P. R. 1990. "Creativity as Creative Thinking." *Music Educators Journal* 76 (9): 22–28. https://doi.org/10.2307/3401073.

Webster, P. R. 2002. "Creativity and Music Education: Creative Thinking in Music: Advancing a Model." *Creativity and Music Education* 1 (33): 16–34.

Weidner, B. N. 2018. "Content Area Literacy in Ensemble Music Education: The Before-during-after Instructional Framework." *Journal of Music Teacher Education* 27 (3): 10–23. https://doi.org/10.1177/1057083717732512.

Wilensky, U., and S. Papert. 2010. "Restructurations: Reformulations of Knowledge Disciplines Through New Representational Forms." *Constructionism* 17:1–15.

Wilensky, U., and K. Reisman. 2006. "Thinking Like a Wolf, a Sheep, or a Firefly: Learning Biology through Constructing and Testing Computational Theories—An Embodied Modeling Approach." *Cognition and Instruction* 24 (2): 171–209. https://doi.org/10.1207/s1532690xci2402_1.

Wing, J. 2006. "Computational Thinking." *Communications of the ACM* 49 (3): 33–35. https://doi.org/10.1145/1118178.1118215.

Wing, J. 2008. "Computational Thinking and Thinking about Computing." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366 (1881): 3717–3725. https://doi.org/10.1098/rsta.2008.0118.

Wing, J. 2011. "Research Notebook: Computational Thinking – What and why." *The Link Magazine* 6:20–23.

Zhang, Y., D. Lusa Krug, C. Mouza, D. C. Shepherd, and L. Pollock. 2022. "A Case Study of Middle Schoolers' Use of Computational Thinking Concepts and Practices during Coded Music Composition." *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education* 1: 33–39. https://doi.org/10.1145/3502718.3524757.