Integrating Computer Science into Music Education

John Peterson
Western State Colorado University
peterson@western.edu

Greg Haynes
Western State Colorado University
grhaynes@western.edu

Abstract

We present our experience in using a Domain-Specific Language, Euterpea, in a general education music class. While the use of *computing* in music education is common, we demonstrate that *coding* allows students without a background in music or computing to explore topics in music form and theory.

Coding supports a new style of music education, one that is focused on composing music rather that performance or appreciation of existing music. We focus on styles of music that can be built algorithmically from a structural description. With such music the use of coding allows students to define and use patterns in a way that makes it possible for complex compositions to be specified in a simple manner. This approach works well in the context of general education: we have designed our curriculum around specific genres of music that are easily represented using simple domain-specific language rather than address all possible topics in music theory through a complex language.

Our experience suggests that a well-designed DSL for describing musical compositions provides a unique way to introduce students to core concepts in music in way that is engaging and pedagogically appropriate for learning about music theory and structure. Students are also exposed to important computing concepts such as language syntax, functions, abstractions, and types. We have also used this approach with pre-college students and believe that this style of music education can be adapted to a K12 environment.

Keywords Domain-specific languages, music education, Haskell

1. Introduction

Computing and music have a long association: programmers are drawn to music synthesis as a way to demonstrate the power of computing in a creative way. Musicians have embraced computers as a new sort of instrument to be used in their craft. More recently, educators have been drawn to the idea that music, and the arts in general, are uniquely motivating to students who are learning about computing.

We approach this issue from the arts perspective, with a goal of bringing concepts in music structure and theory to the front. By using coding as part of the arts curriculum we demonstrate the power computing outside the computer science curriculum, reaching students that would not otherwise be interested in computing.

SIGCSE '17, October 25–30, 2015, Seattle, Washington, United States.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

As our primary goal is to teach music theory and structure, the underlying programming language must directly express the concepts and structure of music. Thus we have chosen a domain specific language (DSL) that primarily consists of music concepts with as little programming language mechanism as possible.

2. A New Approach to Music Education

There are two novel aspects to our class. First, traditional music notation is replaced by a computer language. Students do not need to read conventional music notation to be able to express music. Second, instead of focusing on either appreciation or the mechanics of performance, students learn about music by creating their own code-based compositions based on existing models. While composition-based music pedagogy is nothing new in itself, our presentation of concepts using top-down archetectural constructs allows students to instantly access and edit deep structure in music, without having to analyze and interpret at the note level[2]. As a general education class, we are not obligated to integrate with other parts of the music curriculum, providing the freedom to depart from traditional notation elements and focus on specific styles of music that are easily described in an algorithmic manner.

The learning objectives of our class are primarily related to music theory: using musical models presented in class, we extract meaningful architectural elements and re-introduce them with a generalized code-based representation. These elements include time-based and pitch-based structures.

The course begins with the study of time-based structures such as meters, rhythms, and accent patterns that are common to a breadth of musical styles and can be easily identified in music for non-pitched percussion instruments. Musical models ranging from marching drumlines, taiko drum ensembles, and Brazilian batucadas are presented to demonstrate fundamental ideas concerning meter, pattern repetition, and layering as they relate to the development of a piece of rhythmic music. In a computing context, it is easy to represent repeated patterns and high level structures, allowing students to succinctly define significant compositions.

Once students have mastered these time-based elements in Euterpea, we introduce pitch-based structures in music. These are presented using both existing music models and code-based representations. Such elements include scale collections, chords, and ideas of functional harmonic progression that correspond to traditional elements of western, common practice, music theory. By combining both time and pitch-based elements within the context of musical phrases and formal sections, students are able to create their own compositions.

Our class alternates between introducing music-based concepts, presented as listening examples and keyboard-based demonstrations, and computational elements that allow students to directly experience music based on these concepts. These concepts then form the basis for creative assignments and are demonstrated in the student compositions. Over the course of the semester, we present

five different creative projects which cover a broad set of music concepts. These projects are discussed in detail in section 4.

In this class, students use a simplified version of Euterpea to develop an understanding of music through the process of composition. For each unit in music and computing, several musical instances and models are presented along with representations corresponding to musical structure and analysis. These structural representations come from current pedagogical techniques in music theory and utilize the same terminologies one would experience in a course devoted exclusively to the study of music theory. The structural representations, including aspects of form, harmony, and motivic structure, are then realized as instances in a functional programming language, rather than in the syntax of music, which would require a substantial emphasis on fundamental music reading techniques. Student compositions can then incorporate these pitch-based ideas along with their time-based structures. Musical models ranging from folk and popular to classical music are used to introduce concepts associated with melodic structure and functional harmony to the class in the same way these ideas are represented in the pedagogy of music theory.

3. A Domain Specific Language for Music

This work is based on an existing domain-specific language, Euterpea[4], to capture the structure of music. This language was developed at Yale as part of their Computing and the Arts program and is embedded in the Haskell language. We have created a simplified version of Euterpea and added vocabulary that captures musical structures, such as notes, scales, chords, rhythms, and harmonic progressions. The purely function nature of this language means that programs are easy to understand by students unfamiliar with the sort of step by step computational process used by conventional programming languages. A type system ensures that programs are correctly formed and there are few run-time errors that are not caught by the type system.

Euterpea is an example of an *embedded* domain-specific language. While the underlying syntax and semantics come from Haskell, most of the vocabulary needed to create musical compositions comes directly from terminology that is already part of music education. Our goal is to minimize the use of computer science related terminology, favoring existing vocabulary. For example, although dynamic levels are represented by numbers at the midi level, we use terms such as mp or fff when possible.

In this class, Euterpea replaces conventional music notation. Students do not need to learn standard music notation since Euterpea code is capable of expressing the aspects of music that are essential to the learning goals of the course. For example, if a homework asks students to build a major scale on a particular note, the answer is expressed using code rather than on a staff.

3.1 Basic Music Vocabulary

Programs in Euterpea are a series of mutually-recursive definitions. A set of built-in names provides the basic music vocabulary. For example, the name `c3` refers to the note C in the third octave. Music can be combined in two ways: sequentially, using `+`, and in parallel, using $*$. Thus the following defines a three note sequence:

```
phrase = c3 + d3 + e3
```

To run a program such as this, the `play` function can be entered at the command line. Thus, once this program is loaded to hear this music a student would type

```
> play phrase
```

One of the basic ideas of simple Euterpea is that defaults are used to make musical objects "complete" by default. For example, the `c3` note has a standard dynamic (mf), duration (whole note), and instrument (piano). These can be altered but the use of defaults ensures that music is always playable. A set of functions that modify music values allows adjustment of these defaults. For example, the `q` function makes music go 4 times faster, turning the default whole note into a quarter note. Played in quarter notes, the previous phrase becomes

```
phrase = q c3 + q d3 + q e3
```

or even

```
phrase = q (c3 + d3 + e3)
```

Note that in Haskell there are no parenthesis needed around function arguments. This would read q(c3) in most other languages. There are many ways to modify a music value: you can play it faster, slower, louder, softer, or on a different instrument. All of these correspond to functions in the Euterpea library. For example, the following plays the above phrase loudly on a marimba:

```
loudm = ff (marimba phrase)
```

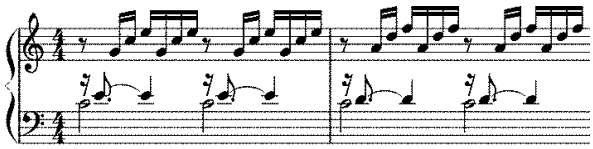
Music contains both pitched sounds, like the notes used above, and unpitched sounds such as a drum strike or cymbal crash. These are represented using names such as `openHiHat` or `lowBongo`. Rhythms are expressed by rhythm strings. This repeats a piece of music according to timing and accent patterns. For example, this plays two rhythms of eighth note sounds:

```
rexample = rhythm "! * > * " (e lowBongo) *
           rhythm " * * * *" (e hiBongo)
```

The `!` and `>` characters represent accented and strongly accented strikes. The vocabulary of Euterpea reflects the elements of a performance that are significant musically - with rhythms, the accent pattern is just as important as the timing for a performance.

3.2 Function Definition

Patterns of music are represented using functions. For example, consider the structure of the Bach C Major Prelude:



Each measure of this music follows the same pattern; given the five notes, the following function creates an entire measure. Also, note that it is not really necessary to understand music notation to see this structure.

```
bach n1 n2 n3 n4 n5 =
     h n1 *
    (s r + (tie de q) n2) *
    (e r + s n3 + s n4 + s n5 +
     s n3 + s n4 + s n5)
```

The `h`, `q`, `e`, and `s` functions correspond to standard note durations: half, quarter, eighth, and sixteenth. The `de` is a dotted eighth and the `tie` function joins two durations. A rest is represented by `r`. Aside from the `+` and $*$ operators, all of the names used (except the parameter names `n1` - `n5`) come from music rather computing principles. This would not be the first example of functions presented but it is typical of the sorts of abstractions that students were capable of once they became comfortable with the Euterpea.

3.3 Higher Level Music Objects

Scales and chords are formed by restricting notes to some subset of the set of all notes. Euterpea considers these to be infinite

sets, although as pitches become too high or low these notes are no longer audible. These sets allow Euterpea to capture common patterns in the music we are familiar with. Within a pitch collection, we can refer to individual notes by number (in music theory this is termed the scale degree). Within a pitch collection, we choose one note to be the center with degree 0. Euterpea has a set of built-in pitch collections that correspond to common pitch collections in music. For example, `majorScale` or `minorTriad`. The melody function creates a melody from a pitch collection, a list of scale degrees, and a list of durations. When the list of durations is shorter than the list of scale degrees it is repeated. This would be a C major scale:

```
cmScale = melody (majorScale KeyC # c3)
                 [0,1,2,3,4,5,6,7,8] [1/4]
```

The # is used to express centering; the scale starts on the note c3. The 1/4 indicates that quarter notes are used to play the scale. The list of numbers that represent scale degree or rhythm can be manipulated in common functional programming style to represent concepts such as diatonic transposition or rhythmic augmentation.

3.4 Full Compositions

Compositions are expressed using base definitions that express small pieces of music, usually a measure long, and grouping definitions which combine these measures into larger structures. Reuse is accomplished through naming. The following program defines a full composition that consists of an 8 chord progression, arpeggiated one measure at a time. Against this background is a simple melody. Chord progressions are one the topics covered in the class and these particular chords were generated by the student from a state machine that describes common practice familiar music. The composition plays the background once before adding the melody.

```
cd1 = majorTriad KeyD # c3
cd2 = majorTriad KeyG # d3
cd3 = minorTriad KeyE # e3
cd4 = majorTriad KeyA # c3
cd5 = minorTriad KeyB # b3
cd6 = minorTriad KeyE # b3
cd7 = majorTriad KeyA # c3
cd8 = majorTriad KeyD # d3

-- A 1 measure arpeggiation
tyarp c = melody c [0,1,2,1,3,1,2,1] [1/8]

-- The harmonic structure
tychords = tyarp ty1 + tyarp ty2 + tyarp ty3 +
           tyarp ty4 + tyarp ty5 + tyarp ty6 +
           tyarp ty7 + tyarp ty8

tymel = dh d4 + q cs4 + b3 + (tie h e) e4 +
        e d4 + e cs4 + e b3 + a3 + b3 +
        dh d4 + e cs4 + e b3 + h a3 + h cs4 + d4
tym = tychords + tychords ∗ violin tymel
```

The design of this library is not arbitrary; all of the features demonstrated above are there to serve specific curricular objectives. This library contains just enough functionality to support specific projects. By presenting code examples with each project, students are able to understand how the code above related to music principles and then modify this code to create music in their own way. This should also demonstrate that students are exposed to a number of important ideas in programming, such as naming, abstraction, lists, strings, and recursion.

4. Projects in Music Composition

In addition to basic tonal music structures such as chords and scales, the curriculum includes many interesting theoretical and structural aspects of twentieth century music that correspond well to essential concepts in computing. Masterworks from various contemporary compositional styles are presented that illustrate effective musical expression using different formal and mathematical structures. Students learn how these systems can be abstractly represented in logical expressions that are manipulated to create different musical instances. Another example includes using code to shift metrical emphases within a series of notes or using incremental time-based functions to create the kinds of phasing manipulations that occur in a piece like Steve Reich's *Clapping Music* or *Piano Phase*. The curriculum exposes students to a wide variety of music and makes each style accessible by allowing experimentation within compositionally appropriate parameters. College students without a background music or computing have been able to master both the musical and computational aspect of this curriculum, composing model-based quality music using a computational process. We believe that this curriculum could be equally effective in a K12 environment.

In this section we present the creative projects used in our class. For each of our projects we discuss the type of music studied, models of music that the students listen to, the underlying music theory, and the computing topics that are covered.

4.1 Rhythmic Music

We start the course with several models representing rhythmic music using unpitched instruments, including excerpts from Japanese Taiko and Brazilian street percussion performances. This music is visually diagrammed in a way that represents form and structure graphically, priming the students to understand how to create similar musical structures using code.

This assignment addresses basic computational skills: using named definitions to express repeated structure, the representation of rhythm and accents using strings, and the use of functions to modify the tempo and dynamics of a composition.

Students are encouraged to group the rhythm strings so that the relationships between different parts become visible, as in a conventional music score. For example, a short composition might start with a set of basic rhythms:

```
rh1 = ">   >   >   >   "
rh2 = "****    ****    "
rh3 = "> * > * > * > * "
```

Note that these string are all 16 characters long, corresponding to sixteenth notes in a 4/4 bar of music. Next, these rhythms are assigned to instruments.

```
bd = rhythm rh1 (s bassDrum1)
lc = rhythm rh2 (s lowConga)
ph = rhythm rh3 (s pedalHiHat)
```

Finally, parts can be layered in, entering one at a time:

```
composition = bd + bd ∗ lc + bd ∗ lc ∗ pc
```

Using this methodology, the student can see the musical architecture in the code and manipulate various parameters to experience how they affect musical form and other aesthetics. It is easy to reuse strings like this for different purposes throughout a composition.

Students learn about naming and reuse, use of strings to represent rhythms, concepts related to meter and rhythm in music, general elements of musical form, and the basic rhythmic structure of Euterpea.

4.2 Chord Progressions and Expression

To introduce the second project, students are exposed to a variety of musical models and asked to recognize the pitch-based components of music, in addition to those based in time. These pitch-based melodies and harmonies are demonstrated in a variety of contexts, including American folk music, popular music, and classical music. A simplified, yet pedagogically valid, model of understanding melody as consisting of singly-issued pitches over time supports an understanding that corresponds to the sequential operator (+), while understanding harmony as consisting of simultaneously invoked pitches supports understanding corresponding the the parallel operator (*). While it is noted that, in the western classical tradition, the octave is divided into 12 steps which correspond to the twelve keys on a piano, it is common for a musical piece to utilize only a subset of these 12 pitches while ascribing a status of resolution, or tonic, to one pitch in particular. This is shown to be the case as pitch inventories are taken for the musical models used in class. Thus, the idea of composing within major and minor scales is introduced with respect to the interval anatomy of the scales prior to representing scale collections in Euterpea.

Regarding harmony, fundamental diatonic triads are demonstrated within the context of the musical models and some general tendencies are observed concerning the progression of one chord to another. These tendencies are discussed and used to create a state machine representing a simplified functional harmony chart to guide students in creating their own musical phrases harmonically.

While students learn to create chords from scratch by naming tradic notes individually, they are soon exposed to representations of tradic pitch collections for more elaborate expressions of harmony. Students learn to express these collections and use functions to create a repeated pattern of chordal harmony. Lists of integers are introduced to create block chords or arpeggiated patterns. Euterpea provides a variety of functions to create music from pitch collections. For example, the following code represents an arpeggiation for a particular chord:

```
m = melody (majorTriad KeyC)
      [0,1,3,2,3,4,2,1] [1/16]
```

Once students aurally appreciate the different harmonic textures available to them via functions in Euterpea, they are then asked to experiment with composing a melody in parallel with a previously composed chord progression, within a global pitch collection (or scale). Thus, the instructional method for this project parallels a traditional song writing workshop in many respects. By using the computer as an instrument and representing music with code, however, the students produce music immediately and are able to perceive the structural elements of their own music without specialized lessons in music syntax and analysis.

4.3 Motivic Development

Many musical compositions are structured using short melodies, referred to as motifs, which constantly recur, often in a transformed manner. Specific examples include Bela Bartok's *Chromatic Invention* theme using a code expression that represents aspects of the theme in independent lists.

In Euterpea, we define these motifs using lists of numbers. One list represents scale degrees and the other durations. These lists can then be manipulated via standard functional programming techniques such as mapping to create motivic developments that mimic those used by composers.

As an example, consider the following lists:

```
m1p = [4,4,5,4,2,3]
m1d = [1/4,1/4,1/8,1/8,1/8,1/8]
```

The `melody` function converts these numbers to music, in this case in the key of C major:

```
motif = melody (majorScale KeyC) m1p m1d
```

Transposition moves the motif up or down in the scale. For example,

```
m1pt = map (+ 1) m1p
```

In Haskell, `(+ 1)` is an increment function. This moves the motif up or down within the context of a particular scale. Using a different scale in the call to `melody` would modulate the melody into a new key.

Other transformations are *inversion*, in which the pitch numbers are negated, *retrograde*, in which the lists are reversed, *augmentation*, in which the durations are multiplied by a constant, and *diminuation* in which durations are divided by a constant. *Fragmentation* is implemeneted using list operations to take subsequences out of these lists.

The composition assigment requires students to explore compositions built from one or two motifs using a variety of these functional programming techniques. Using musical models, we identify a variety of time and pitch-based motivic development techniques used by composers that corresspond to the motivic alterations created by the mapping functions.

4.4 Minimalism and Transformational Harmony

To expand an understanding of harmony and contemporary composition, we expose students to musical models that use traditional elements of pitch, meter, and triads, but in a context that does not utilize a scale collection or tonal center. Various works by minimalist composers such as Philip Glass utilize triadic structures that transform and progress within a repeating arpeggiating texture and do so without reference to a tonal center[5]. Thus, the chords are related to one another by rules that can be expressed in functional programming. These rules are bi-directional and may be described as follows: p transforms a major chord into a minor chord based on the same root, r transforms a major chord into minor chord based on the root of the relative minor, and l transforms a major chord into a minor chord based on the root of the 3rd scale degree. The rules may be more easily expressed in the following chart:

```
P: Parallel ex. C to cm
M: third moves down a half step
m: third moves up a half step

R: Relative ex. C to am
M: 5th moves up a whole step
m: root moves down a whole step

L: Leading-tone exchange ex. C to em
M: root moves down a half step
m: fifth moves up a half step
```

The three transformations, p, l, and r, are represented by functions on pitch collections. Function composition allows multiple transformations to be applied in a single step. This defines a set of four chords drived from these transformations:

```
ch1 = minorTriad KeyA # a3
ch2 = tl ch1
ch3 = tr ch2
ch4 = (tp . tl) ch3
```

As with the assignment in chord progressions, these pitch collections can then be elaborated into larger structures involving arpeggiation, block chords, or even multiple simultanous arpeggiations to create interesting harmonic expressions of the chords.

In addition to just creating music that uses this transformational harmony, students are exposed to Philip Glass's film score for *Koyaanisqatsi* to open discussion regarding collaborative art pieces. Students investigate the relationship between film and music. It is demonstrated how changes in tempo, dynamics, and other expressions correlate to the events and transitions in the film. The full creative assignment entails students selecting video excerpts listed under a Creative Commons license and compiling short films for which they compose scores utilizing the transformational harmonic techniques and arpeggiation functions.

4.5 Chance and Chaos

To further expand the students experiences with music and code, several contemporary musical pieces are modeled that utilize either chance or indeterminacy as a compositional device. One of these pieces includes Terry Riley's *In C*. This work consists of an ordered list of motifs for which each individual within an ensemble of players is instructed to repeat the first motif an indeterminate number of times before moving on to the next figure, and so on. The resulting music may be described as a globally controlled sound mass; the piece may be described in general terms, but each individual performance is quantitatively unique.

To express chance music (also called stochastic music) Euterpea uses seeded randomness. For example, the choices function uses a random seed and a count to create a list of values taken from a list. In the following, the line function takes a list of music values and plays them sequentially.

```
randomNoteList = choices 123 10 [c3, e3, g3]
randomComposition = q (line rnotes)
```

More structured randomness can be obtained through a Markov chain, which is created in a straightforward manner from a functions provided in Euterpea.

Within this instructional unit, other models of music are presented that utilize nonmusical mathematical constructs for musical purposes, such as pieces by Iannis Xenakis and Milton Babbitt. Thus, in addition to ordinary randomness, students use chaotic number sequences, such as the tinkerbell chaotic map, to generate structured musical objects. Students adjust parameters within the representative functions to create mapping changes between the chaotic sequence and musical elements, thus affecting interesting aesthetic change.

5. Experiences

5.1 College Level General Education

This work is the result of a collaboration between faculty in the Computer Science and Music departments. We were able to insert Algorithmic Music into our institutions general education progam (as a music class rather than a class in computing). This presented an opportunity to test our curriculum out on a cross section of students at our institution. Some of these students had a background in music or computing but the majority of them came into the class with no knowledge of either.

Students found the course to be engaging and creative. Evaluations were uniformly positive and much higher than those in other general education music courses. We were suprised by how little help the students needed in coding. Once they learned how to read error messages they were generally able to resolve program errors without assistance. Compositions produced in this class were typically 30 to 50 lines of code.

One of the main issues in teaching a course based on creative design is how to assess student efforts. We did not want to give subjective grades that reflect our personal biases as to what is good or bad in music. Rather, we based grades on a combination of objective measures, how appropriately structures are used and how well they correspond to existing musical models, and a documentation of the creative process. Students were expected to submit multiple versions of each work and explain their goals and revisions. We found that there was a clear correlation between their ability to document their creative efforts and the quality of the work they produced.

We found that the music department was very supportive of our efforts. Many of the music faculty attended a concert of works produced in this class and commented favorably on the musical qualities of the compositions.

5.2 Middle School Outreach

This software was used in an outreach program for middle school students. As in the class, students found the material engaging. The younger students required more assistance than the college students had but older students (those about to enter 8th or 9th grade) were able to handle the software with relatively little assistance. With improvements to the software and better classroom support we conjecture that this would be an excellent way to bring computing into a middle or high school environment.

6. Related Work

There are many excellent libraries for expressing music in various programming languages, particularly Python. Systems such as Jython Music[6] and Earsketch[3] facilitate the use of music as part of a computing curriculum. However, these libraries lack the music level abstractions of our system and require a much deeper study of computing.

This effort is a microcosm of a larger movement, the development of academic programs in computing and the arts. A number of schools have developed interdisciplinary programs that mix computing and music, including Yale and Charleston[7]. Indeed, Euterpea was developed to serve the Computing and the Arts program at Yale. Our goal is to replicate this sort of experience in a single semester class.

Introducing students to music through composition is not a new approach[1]. However, existing practice is commonly based on a performance-oriented music representation, as entered through a midi keyboard or sequencer software. Our system differs from these in that the student deals directly with the music structure rather than the performance, making it harder to express patterns and placing more complex musical styles out of reach.

Many professional music authoring programs have a scripting language that is capable of generating complex music using code. Examples of this include production software such as Reaper, which includes a code-based feature called ReaScript. However, these languages are not suitable for use by novices. Experience with our system, though, should help students understand these tools if they choose to pursue music production or algorithmic composition in a professional context.

7. Future Work

The programming environment for Euterpea is the biggest roadblock to adoption of this system. At present, students interact with the Haskell compiler, ghci, from the command line. Students do most of the debugging needed in the type checker - better error messages and interaction would make the task of coding much easier. At present a browser-based version of our system that does not need to support the full Haskell language is under development.

There are many other styles of music that can be expressed in an algorithmic manner - we would like to include more on them in future versions of this class and add support for other compositional styles to Euterpea.

As this project matures, formal assessment of the effectiveness of our curriculum on topics in music and computing is needed. In particular, to move this work into a K12 environment we need to be able to quantify the learning experiences of students.

We plan to support this curriculum at other institutions. In addition to our software, we are also creating a website with documentation, examples, and sample assignments for other instructors to use. We hope to provide enough material that music faculty can teach this course without the help of faculty in computer science.

8. Conclusions

Our work demonstrates a productive collaboration between computing and the arts. Computing provides a new way to deliver an existing arts curriculum while the arts provide a problem domain in which students experience the power of computing. Although students did not regard this material as a "computer class", they were exposed to significant topics in computing, including programming language syntax, naming and reuse, functional abstraction, and type systems.

By minimizing the "accidental difficulties" introduced by the programming language, we believe that this curriculum can be delivered by instructors without extensive training in computing and that Euterpea can be used in a variety of teaching environments, from middle school to college, as a vehicle for music instruction.

Our work also brings up the bigger picture - how will computing move from a specialized skill to a universal competency such as math? By incorporating coding into the music curriculum, we show one possible path for computing to become integrated into other disciplines without requiring students to use the step by step programming model of traditional programming languages.

Acknowledgments

This work is supported by NSF Grant 1302327.8

References

[1] T. S. Brophy. Building music literacy with guided composition. *Music Educators Journal*, 83(3):15–18, 1996. ISSN 00274321, 19450087. URL http://www.jstor.org/stable/3398972.

[2] B. Freedman. *Teaching music through composition: A curriculum using technology*. Oxford University Press, 2013.

[3] J. Freeman, B. Magerko, and R. Verdin. Computer science principles with earsketch (abstract only). In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 710–710, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-2966-8. . URL http://doi.acm.org/10.1145/2676723.2678282.

[4] P. Hudak, D. Quick, M. Santolucito, and D. Winograd-Cort. Real-time interactive music in haskell. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Functional Art, Music, Modelling and Design*, FARM 2015, pages 15–16, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3806-6. . URL http://doi.acm.org/10.1145/2808083.2808087.

[5] D. Lewin and S. Rings. David lewin. *Journal of Music Theory*, 50(1), 2006.

[6] B. Manaris and T. Kohn. Making music with computers: Creative programming in python (abstract only). In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 711–711, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3685-7. . URL http://doi.acm.org/10.1145/2839509.2844707.

[7] B. Manaris, R. McCauley, M. Mazzone, and W. Bares. Computing in the arts: A model curriculum. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 451–456, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2605-6. . URL http://doi.acm.org/10.1145/2538862.2538942.