EarSketch: An integrated approach to teaching introductory computer music

SCOTT MCCOID*, JASON FREEMAN*, BRIAN MAGERKO[†], CHRISTOPHER MICHAUD[‡], TOM JENKINS[†], TOM MCKLIN[§] and HERA KAN[¶]

*Georgia Tech Center for Music Technology, 840 McMillan St, Atlanta GA 30332-0456, USA
[†]School of Literature, Media & Culture, Georgia Tech, 686 Cherry St. Atlanta, GA, 30332, USA
[‡]Marist School, 3790 Ashford Dunwoody Road, NE, Atlanta, GA 30319-1899, USA
[§]The Findings Group, 1201 Clairmont Road, Suite 305, Decatur, GA 30030, USA
[¶]Eliot-Pearson School of Child Development, Tufts University, 105 College Avenue, Medford, MA 02155, USA
E-mails: smccoid3@gatech.edu; jason.freeman@gatech.edu; magerko@gatech.edu; michaudc@marist.com; tom.jenkins@gatech.edu; tom@thefindingsgroup.com; hera.kan@tufts.edu

EarSketch is an all-in-one approach to supporting a holistic introductory course to computer music as an artistic pursuit and a research practice. Targeted to the high school and undergraduate levels, EarSketch enables students to acquire a strong foundation in electroacoustic composition, computer music research and computer science. It integrates a Python programming environment with a commercial digital audio workstation program (Cockos' Reaper) to provide a unified environment within which students can use programmatic techniques in tandem with more traditional music production strategies to compose music. In this paper we discuss the context and goals of EarSketch, its design and implementation, and its use in a pilot summer camp for high school students.

1. INTRODUCTION

As the scope of computer music as both an artistic practice and an interdisciplinary research field has rapidly expanded in recent decades, the curriculum in introductory-level survey courses in electronic and computer music has necessarily evolved. Such survey courses have either traditionally aimed to arm music majors with basic literacy as users of commercial sequencing, recording and notation software (Webster and Williams 2006) as articulated by national standards (NASM 2012), or sought to teach a non-specialised group of students (often non-majors) the tools of commercial recording studios and/or MIDI-based production techniques and to challenge them to create their own music in the studio.

In recent years, a growing number of universities have taken a more holistic approach to these survey courses, introducing students not only to the dominant paradigms in commercial music software and their connection to music education and composition, but also to the broader field of computer music research that includes new musical interfaces, interactivity, music information retrieval, algorithmic composition and sonification, among others. These courses also include a basic grounding in electroacoustic music history, theory and analysis. Topics typically include acoustics, psychoacoustics and cognition; audio recording and editing; MIDI sequencing; synthesis and digital signal processing; algorithmic composition; music information retrieval; and spectral analysis and synthesis. Students explore these topics both by composing their own music using a variety of commercial music production tools and by completing simple music-programming assignments. This paradigm is represented by two recent textbooks (Collins 2010; Burk, Polansky, Repetto, Roberts and Rockmore 2011) and by courses at several major universities (Klingbeil 2009; Ballora and Craig 2010; Burtner 2012; Garton and Diels 2012) including our own institution, Georgia Tech.

1.1. Challenges and opportunities for holistic courses

The expanded scope of these courses, coupled with changes in computing technology and the studio paradigm since the early 2000s, have created significant practical and pedagogical challenges that suggest the need for new, integrated software tools. Within these challenges, we also see the opportunity to more directly connect the pedagogy of these courses to introductory computer science education: using computer music to motivate computer science learning (and vice versa) to better motivate students to pursue further studies in either discipline.

In order to adequately address the course's curriculum, classes typically use many different software programs over the course of a semester: at Georgia Tech, for instance, our course requires students to use Ableton Live, Propellerheads Reason and Cycling '74's Max/MSP (and often other programs) to complete assignments. While requiring students to use multiple pieces of software does give them a greater sense of available tools, it places a large burden on them to master a number of programs in a limited amount of time. Equally important, this practice can instill a sense of division between artistic and research curriculum, between composition and programming, and between commercial and academic work. In particular, we have found that many students are unmotivated to complete programming assignments and prefer to spend time using commercial tools: their limited programming skills make it hard for some of them to create satisfying music and sounds within the course's short timeframe, while the commercial tools enable them to quickly create results that provide more instant gratification.

Inter-application communication is, of course, a natural strategy to address the integration problem, but technologies such as inter-application MIDI, ReWire and JackAudio (MIDI Manufacturers Association 2012; Propellerhead Software 2012; Davis 2011) are challenging to configure, making them problematic to incorporate into introductory courses. While Max for Live offers an alluring approach for tight integration, it comes at a price: both the actual cost of the software products themselves (see further discussion below) and the relative disconnect between programming paradigms in Max and those in general-purpose programming languages (which works against the opportunity to integrate computer science pedagogy into the course).

Beyond the complexities of learning and potentially integrating multiple software programs during a one-semester survey course, the use of multiple programs poses practical issues. While most university students have computers, and many universities in fact require students to own laptops (Schedel 2007), students are rarely able to use their own computers to complete coursework. At the introductory level, the barrier is not processing power or outboard hardware, as it once was, so much as the prohibitive cost of software licences: single-user academic licences typically cost US$250 for each product. While open-source alternatives to many programs do exist, those products often have less documentation and a higher learning curve, creating practical problems in instruction and support. And recent trends towards cloud-based virtual machines can solve software licensing issues but are impractical for most music applications due to degraded audio quality and high latency.

Instead, universities usually provide classroom labs and/or private studios in which students can complete coursework on workstations with required software preinstalled. While this shared workspace can provide a sense of community for students, it can also inconvenience them, limit the time they can spend working on assignments, limit course enrolment based on studio resources, and impose an ongoing financial burden on the institution to maintain and upgrade facilities to support these introductory courses.

1.2. The EarSketch approach

In this paper we describe EarSketch, our all-in-one approach to supporting a holistic introductory course in computer music as an artistic pursuit and a research practice. Targeted to the high school and undergraduate levels, EarSketch enables students to acquire a strong foundation in electroacoustic composition, computer music research and computer science. EarSketch integrates a Python programming environment with a commercial digital audio workstation (DAW) program (Cockos' Reaper) to provide a unified environment within which students can use programmatic techniques in tandem with more traditional music production strategies to compose music. Students learn how programmatic approaches to composition help automate and speed up tedious tasks, such as specifying effects automation break points; how they facilitate a rapid iterative compositional approach in which small changes to code can be quickly auditioned; and how they enable the use of novel algorithmic techniques, such as stochastic processes, to create unique sounds, structures and textures. Our curricular materials aim to reinforce this integrated workflow by introducing music production concepts, compositional techniques and computer science principles together. EarSketch enables students to use their own computers to complete coursework by building around a single commercial program (Reaper) that is economical (US$60 for noncommercial use), that runs on both Mac and Windows, and that is well optimised for systems with low memory and slower processors. To make up for the loss of community around a shared studio space, EarSketch integrates a social media website which invites students to upload their music and source code, view other students' work and create derivative musical remixes from other students' code.

In the sections below, we describe relevant background to EarSketch in computer science and computer music pedagogy, describe EarSketch and its implementation in detail, and describe and evaluate its use in a pilot summer camp for high school students.

2. BACKGROUND

2.1. Computer music and music technology pedagogy

Little consensus exists as to the exact pedagogical scope of the field of computer music, though some (e.g. Pope 1994) have attempted to develop thorough taxonomies of the field. In particular, there is a natural divide between practical skills in music technology literacy (e.g. MIDI and multi-track sequencing, recording, mixing and music engraving), technical research areas (e.g. synthesis and signal processing, algorithmic composition and music information retrieval), and electroacoustic composition, theory and analysis (which may draw skills from both of these other areas). Ballora and Craig note that 'one need only compare Dodge and Jerse's *Computer Music* with Webster and Williams' *Experiencing Music Technology* to discover two entirely different sets of goals and assumptions about computers, audio technology, and music' (2010).

Many introductory computer music courses have evolved from focusing on just one of these areas to covering all three. Older computer music texts concentrate mainly on offline synthesis and score-based composition (Moore 1990; Dodge and Jerse 1997). Roads (1996) serves as a more complete reference on all areas of computer music, but, though it is often used as a classroom textbook, its overwhelming size has always made it difficult to use in its entirety within a single introductory course. Collins (2010) and Burk et al. (2011) offer updated, concise and comprehensive introductions to all three areas of computer music. For instance, Collins's text covers areas as diverse as recording and spatialisation, MIDI, music information retrieval and mobile music. All of these are presented without the assumption that readers have access to a studio.

As computer music has developed as a technical research discipline, it has driven an evolution in curriculum and pedagogy; there is a necessity to move beyond the standard studio paradigm. Cipriani and Giri argue that there can be a disconnection between theory and practice, and therefore propose a higher level 'interactive self-learning environment' within Max/MSP for teaching computer music in the context of signal processing and sound design (2011). Additionally, Wang et al. and Essl use a laptop orchestra and mobile phone ensemble, respectively, as a teaching resource, thus moving 'away from strictly studio-oriented computer music courses, instead focusing on live performance' (Wang, Trueman, Smallwood and Cook 2008: 35; Essl 2010).

2.2. Computing pedagogy

Introductory computer science courses 'are not currently successful at reaching a wide range of students' (Guzdial 2003: 104), especially under-represented populations in computing, such as women and ethnic minorities (Eugene and Gilbert 2008). There is a long history of research into building different types of programming environments intended to make computer science more accessible to a broader audience (Kelleher and Pausch 2005). For instance, Rich, Perry and Guzdial developed an introductory computer science course and textbook, *Introduction to Media Computation*, focused on attracting and retaining women by contextualising assignments around creating media. The course uses creativity 'to counter the reputation of CS as boring' and relevance 'to illustrate that CS concepts are applicable to non-majors' (2004: 190). Because students in an introductory course will probably not have existing programming skills, these approaches are intended to bridge the gap while maintaining the students' attention. Students tend to respond well to immediate tangible rewards such as creating (and perhaps sharing) digital media (Malan and Leitner 2007) and digital games (Preston and Morrison 2009). Programming languages and environments have been developed to support these pedagogical approaches, such as Scratch (Monroy-Hernández and Resnick 2008), Alice (Kelleher, Pausch and Kiesler 2007), and JES (Guzdial 2003).

2.3. Combined pedagogy

Finding the right motivation can be key in encouraging an interest in computing among a broad spectrum of students. Particularly with respect to music, the hope is that a large and diverse audience will be more interested in the creation of this type of content (as opposed to programming for its own merits). Hamer describes a particular project used to teach software engineering design patterns, where 'students are encouraged to discover and express their innate musical talents', hoping that the project will 'deepen understanding in programming by exploiting an analogy between program structure and musical structure' (2004: 156). He specifically relates musical elements such as sequential composition, parallel composition, musical repeats and variant endings to programming concepts such as statement sequencing, threaded code, loops and conditional statements, respectively. Ruthmann, Heines, Greher, Laidler and Saulters (2010) build upon these connections between computer science and music by using the Scratch programming environment, which is mainly designed to create interactive media and games, to teach computational thinking through creating music. Students learn important programming concepts such as looping, use of variables and modularisation in a musical context.

Within combined pedagogy settings, many educators are reluctant to use dedicated computer music programming languages and environments. Misra, Blank and Kumar chose instead to use Python, with myro.chuck, as a module in an introductory computing course, because they 'hesitated to confuse novices by teaching in multiple languages' (2009: 248). Many computer music software languages and environments may be 'excellent tools for computer musicians, but have not been designed for pedagogical purposes' (Misra, Blank and Kumar 2009). They argue that, within an introductory computer science context, it is important to teach a general-purpose programming language so that programming skills are immediately transferable to other settings.

These approaches combining computer music and computer science pedagogy address many of the motivational issues in computing education by making abstract computing concepts relevant to students. But they do not necessarily enable students to create music that is stylistically meaningful to them. Heines's and Hamer's approaches rely on event-based composition with MIDI, thus requiring that students possess prerequisite music theory knowledge and limiting the potential population of students. JES provides low-level sample access to audio, so students focus more on extremely basic audio synthesis and manipulations than on creating music. myro.chuck abstracts audio synthesis and audio manipulation to a higher level, such that students do not manipulate individual samples, but students still compose at the note/event level representation. Within an introductory course setting, we believe there should not be any musical or technical prerequisites to creating personally relevant and satisfying music.

3. GOALS

We intend EarSketch to serve both practical and pedagogical goals in the context of introductory courses in computer music that have no musical or technical prerequisites. On a practical level, we want EarSketch to make it possible for students to use their own computers to complete coursework, purchasing a single commercial program that costs less than most course textbooks. This should enable institutions to offer such courses in larger formats and with fewer dedicated resources. Given that demand for these courses usually far exceeds availability, this is an important step towards expanding access to computer music education and ultimately to growing the discipline.

We expect the use of a single, integrated software environment to have practical and pedagogical benefits for students. EarSketch offers a unique paradigm for algorithmic composition that combines functionality for analysis, synthesis and multi-track timeline editing within the context of a full-featured DAW (Reaper) and programming language (Python). With less time spent learning multiple software packages and struggling to integrate them, users can focus more on the core artistic and technical challenges of computer music. Equally important, we believe that the tight integration of programmatic and production-oriented work in a single environment will motivate the pursuit of both and offset the potential motivational problems caused by the differential complexity of musical results that beginning students normally achieve with each approach. Ultimately, we hope that the benefits of this environment extend beyond pedagogy and that EarSketch's unique integration of components will facilitate new approaches to musical expression for composers at all levels.

Finally, we believe that the integration of a programming language popular both in the computer science industry and in education (Shannon 2003; Sloan and Troy 2008) provides a strong gateway between computer science and computer music, especially when the connections between computer science principles, compositional techniques and computer music concepts are emphasised in curricular design. This approach can provide the initial impetus for students interested in computer music to transition into studies and careers in computer science, and for avid computer programmers to find their way into computer music.

4. DESIGN AND IMPLEMENTATION

4.1. Technical design

EarSketch is built on top of Reaper, an intuitive DAW program comparable to those used in professional recording studios. Reaper supports many of the techniques covered in an introductory course, such as multi-track and MIDI sequencing, and it includes 230 built-in effects such as reverb, compression and multi-band equalisation. Its graphical user interface (GUI) resembles common DAW software, such as Logic, Pro Tools and Cubase (see Figure 1). We chose Reaper because of its low cost, cross-platform capability and ability to run on a wide range of computer processor speeds. EarSketch integrates three components into the Reaper software: an application programming interface (API) for controlling Reaper; a sound library of precomposed audio loops; and utility scripts to seamlessly integrate with our social network website, sound library browser and text editor.

Rather than using existing computer music software environments, EarSketch uses the Python programming language for practical (integration with Reaper) and pedagogical reasons. Practically, the EarSketch API leverages ReaScript, Reaper's own Python API, which provides low-level programmatic access to Reaper. Pedagogically, Python is being increasingly used in introductory computing courses (Shannon 2003) and is the eighth most popular programming language worldwide (Tiobe Software 2012). Additionally, Python contains a rich standard library and supports multiple programming paradigms (Nielson and Knutson 2004).

The ReaScript API offers access to much of Reaper's core functionality, including importing and placing audio clips, adding and configuring effects, creating envelopes to automate changes over time and accessing sample data from audio tracks. While the ReaScript API is powerful, it is also difficult to use, with minimal documentation and cryptic

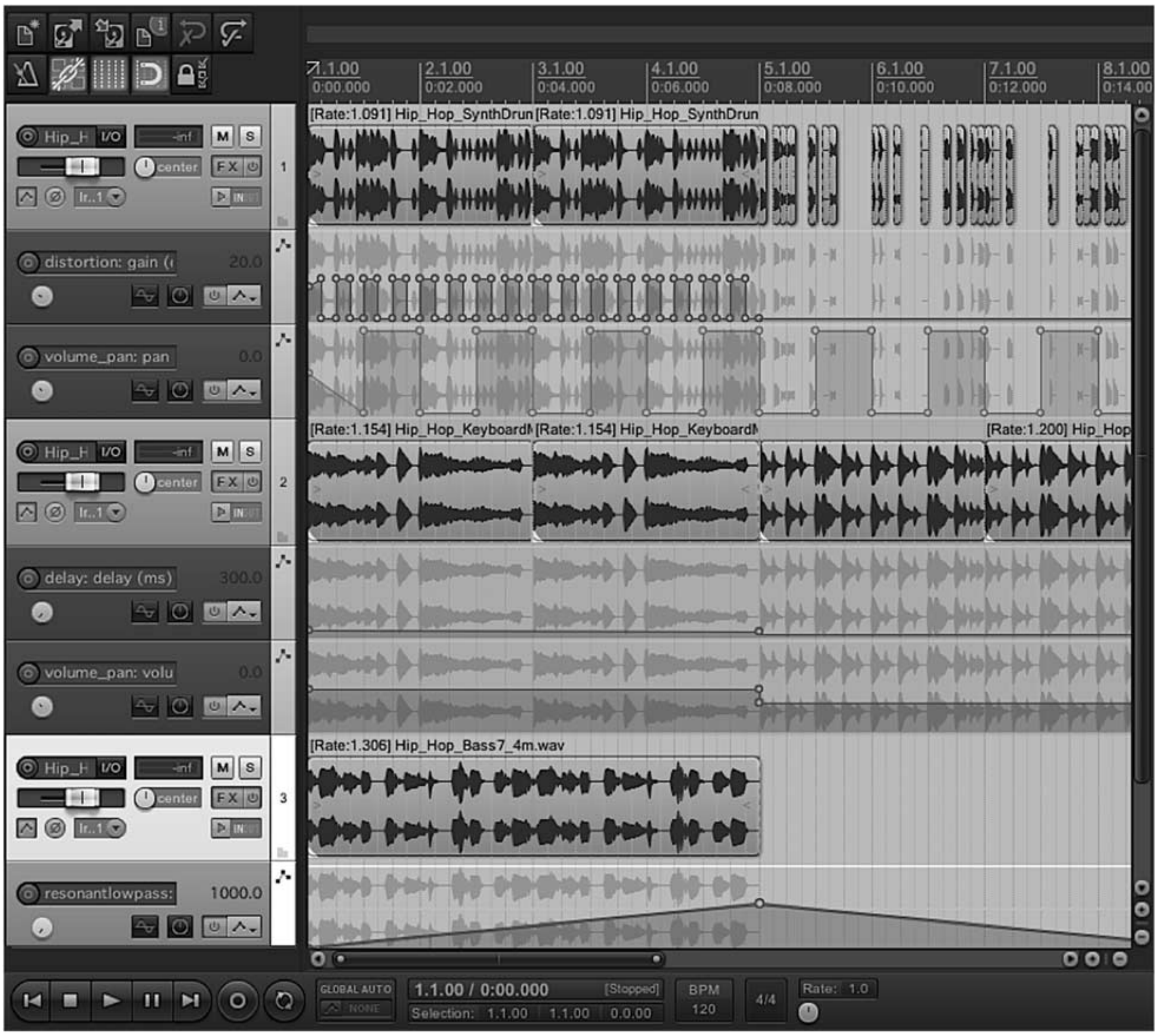


Figure 1. Reaper's graphical user interface.

function calls. We have developed a cleaner, higher-level Python API for pedagogical use that wraps the ReaScript API. It abstracts operations that would require numerous low-level function calls into single function calls specifically focused on composition tasks, such as placing audio files on the timeline, applying effects and creating rhythmic phrases. Students import our API as a Python module and write their script using our functions. Figure 2 compares importing an audio file on track 1, starting on the second beat of the third bar, with the ReaScript API directly and with the EarSketch API. The ReaScript API requires more code, using low-level pointers, manually inserting tracks, and moving the cursor (playhead) manually, to accomplish the same task.

At its essence, the EarSketch API provides functions for placing audio files and effect automation breakpoints on the timeline within Reaper. The API uses a floating-point number time representation, where the integer part (left of the decimal place) represents bar numbers and the fractional part (right of the decimal place) represents the percentage into that bar. For example, in 4/4 time, the number 2.5 represents the third beat of the second bar. The third beat occurs at 50 per cent, or halfway into the second bar. This approach works well for students unfamiliar with metrical formats in Western notation, lends itself easily to mathematical manipulation and logical comparison, and is common in other computer music programming environments, such as JMSL (Didkovsky and Burk 2004).

Figure 3 presents a basic EarSketch script that imports the EarSketch Python module, initialises EarSketch, sets tempo and places an audio file on the timeline. EarSketch can place any audio file onto the timeline by referencing its absolute path on the

```
# Insert an audio file at designated track and location using Cockos' ReaScript Python API
RPR_InsertTrackAtIndex(trackNumber, True)
RPR_Main_OnCommand(40769, 0)
mediaTrackPointer = RPR_GetTrack(0, trackNumber)
RPR_SetTrackSelected(mediaTrackPointer, True)
measure = int(location)
remainder = location - int(location)
subMeas = 4.0 * remainder + 1.0
beat = int(subMeas)
subDiv = 4.0 * (subMeas - beat) + 1.0
sdTimeTuple = RPR_TimeMap2_beatsToTime(0, 1, 0)
absTimeTuple = RPR_TimeMap2_beatsToTime(0, beat - 1, measure - 1)
absTime = absTimeTuple[0] + ((subDiv - 1) * (sdTimeTuple[0] / 4))
RPR_SetEditCurPos2(0, absTime, 0, 0)
RPR_InsertMedia(fileName, 8)


# insert an audio file at designated track and location using the EarSketch API
insertMedia(fileName, trackNumber, location)
```

Figure 2. Comparison between importing an audio file using the ReaScript API and using the EarSketch API.

```
from earsketch import *

# initialize Reaper
init()
setTempo(120)

# Add a music file to track 1
insertMedia(DRUM_N_BASS_DRUMS2_1M, 1, 1.0)

finish()
```

Figure 3. Example of a basic EarSketch script.

file system. The software comes bundled with over 400 commercial audio loops in a variety of popular music styles: hip hop, soul, R&B, techno, house, drum 'n' bass and Latin/world. This content, which we licensed from a commercial vendor, is encoded as acidised WAV files (Grosse 2008) that automatically scale to the project's current tempo and can be referenced within Python through pre-defined constants, to avoid the need to use absolute file paths. (If users place their own audio content into a designated folder, then constants are automatically created to reference their audio files as well.) We focused on popular musical genres with the bundled content in order to match the musical tastes of a majority of our students, but of course EarSketch can be used to create music in any style.

Our API also provides functionality for inserting audio effects on specified tracks and setting automation breakpoints for them. Reaper includes a large number of built-in audio effects. These effects range from traditional delay-based effects to more esoteric spectral transformations. We included predefined constants for students to easily access each effect via the API, including: 3-band equaliser, band-pass filter, chorus, compressor, delay, distortion, flanger, low-pass filter, noise gate, panning, parametric equaliser, phaser, pitch-shift, reverser, ring modulator, scratchy (auto-scratch effect), stereo delay, sweeping low-pass filter, tremolo, variable length delay, volume, wah and waveshaper. Using the setEffect() function, students specify an audio effect and track number, along with start point and end point locations. Figure 4 presents code using the setEffect() function, which was used to generate the automation in Figure 1.

We have also used ReaScript to customise the functionality of Reaper's user interface, hiding extraneous GUI items and adding menu commands to run and edit Python scripts, upload projects to the social media site, access EarSketch curricular materials, and browse and preview bundled audio content. To run an EarSketch script, students select a menu option in Reaper to run a script and then choose the script from a file browser. The results are rendered into a new Reaper project that students can immediately see, hear and further edit in Reaper's GUI. Error messages, if any, are displayed in pop-up dialog boxes or a console window within Reaper.

4.1.1. Sequencing

The EarSketch API does not directly support MIDI sequencing (though Reaper's GUI does), since, in keeping with our goals, we wanted students with no formal musical background to be at ease using the environment. However, EarSketch does support a form of step sequencing through programming, encouraging students to create their own rhythmic material by piecing together contents of different audio files at a semiquaver resolution.

```
from earsketch import *

init()
setTempo(120)

music = HIP_HOP_KEYBOARDMELODY2_2M
drums = HIP_HOP_SYNTHDRUMS1_2M

def distortionRhythm(track, measure):
    for count in range(4):
        sub = measure + count/4.0
        setEffect(1, DISTORTION, DISTO_GAIN, 25, sub, 25, sub + 0.125)
        setEffect(1, DISTORTION, DISTO_GAIN, 0, sub + 0.125, 0, sub + 0.25)

def panRhythm(track, measure):
    setEffect(1, VOL_PAN, VOL_PAN_PAN, -100, measure, -100, measure + 0.5)
    setEffect(1, VOL_PAN, VOL_PAN_PAN, 100, measure + 0.5, 100, measure + 1)

fitMedia(drums, 1, start, middle)
fitMedia(music, 2, start, middle)

for measure in range(start, middle):
    distortionRhythm(1, measure)
    panRhythm(1, measure)

finish()
```

Figure 4. Example of creating effects automation envelopes using the EarSketch API.

```
from earsketch import *

init()
setTempo(152)

beatElements = [DRUM_N_BASS_DRUMS1_4M, DRUM_N_BASS_DRUMS2_1M, DRUM_N_BASS_DRUMS3_1M]
beatString1 = "0---0+0+00+--0--"
beatString2 = "0+--1+++0+--1---"

makeBeat(beatElements, 1, 1.0, beatString1)
makeBeat(beatElements, 1, 2.0, beatString2)

finish()
```

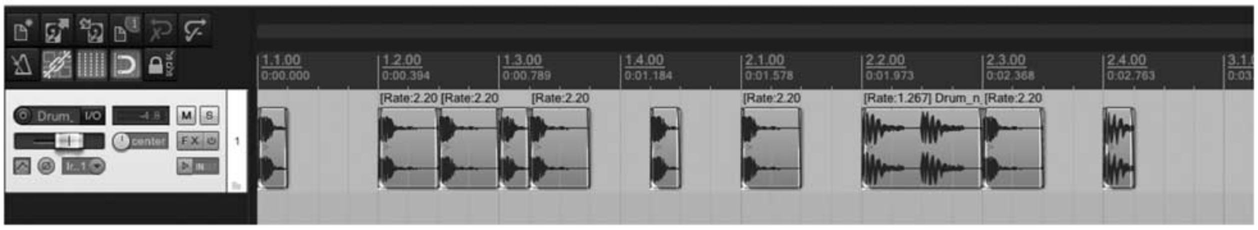


Figure 5. A code example of step sequencing and its output in Reaper.

Inspired by Thor Magnusson's ixi lang (Magnusson 2011), our API uses a similar string representation to specify note locations and durations. Like ixi lang, numbers represent a sound – in our case, a sound file or segment of a sound file at a semiquaver duration. Unlike ixi lang, which uses space characters, appending a '+' sign extends the duration of the preceding sound by a semiquaver and inserting a '-' sign at any point signifies a rest. Figure 5 demonstrates this syntax and its output in Reaper.

Because EarSketch represents these beats as strings, students can manipulate them using standard Python string operations; this teaches string operations in computing in tandem with serial, stochastic and other approaches to the manipulation of musical events. We provide our own functions for common manipulations – reverseString(), shuffleString(), and replaceString() – so that students can begin this experimentation before they learn string manipulation techniques in Python in depth.

4.1.2. Analysis

Not only does EarSketch provide functionality for placing audio content on the timeline, it also provides functions to analyse audio content. We provide the ability to calculate common features – spectral centroid, RMS amplitude, spectral rolloff, spectral flux,

```
from earsketch import *

# initialize Reaper
init()
setTempo(120)

# set up my parameters for this run
track1loop = ELEKTRO_HOUSE_DRUMS1_2M
analysisMethod = RMS_AMPLITUDE
hop = 0.0625  # analyze in 1/16th note chunks
start = 1
end = 3

# build a list of tuples with analysis values with corresponding index
location = start
index = 0
analysis = []

while location < end:
    val = analyzeForTime(track1loop, analysisMethod, location, location + hop)
    analysis.append((val, index))
    location += hop
    index += 1

# sort the analysis list
analysis.sort()

# put it back together shuffled
for index, chunk in enumerate(analysis):
    location = start + chunk[1] * hop
    insertMediaSection(track1loop, 1, start + index * hop, location, location + hop)

# we're done
finish()
```

Figure 6. Example script using an analysis function to sort segments of an audio file from lowest to highest RMS value.

spectral decrease, spectral flatness, spectral irregularity, spectral kurtosis, spectral skewness, spectral slope, spectral spread and zero cross rate, which are typically used in the Music Information Retrieval (MIR) community – and to use that information in a compositional context. This approach follows from projects such as MEAPSoft (MEAPSoft 2008), but with a focus on students making decisions through programming about exactly what to analyse and how to use the results of analysis in their work, rather than choosing from a limited set of algorithms through a graphical interface.

Pedagogically, analysis is used to introduce concepts such as conditional logic in computer science and data mapping in computer music. More broadly, it introduces the notion that, in both compositional and interactive contexts, computers can extract information from audio and can use that information to drive musical decisions (a foundational concept in both computer music research and machine learning). Assignments supporting this pedagogy might ask students to gate tracks to turn their volume up only if an analysis feature goes above (or below) a threshold value, to place on the timeline the audio clip from a list with the maximal (or minimal) feature value, or to map (with scaling) a feature value to an effect automation parameter. (This component of our curriculum will be piloted in summer 2013.)

So that students can quickly explore different features for analysis, we scale all the feature values between 0 and 1, and limit the results to a single scalar value. We believe this will allow students to experiment with different features within their composition and eventually gain more of an intuition as to how these features correspond with the audio content. Students do not lose any precision by using scaled values, and therefore they can apply additional scaling to these values to use in conjunction with effects or other operations. Unfortunately, not every useful feature calculates a scalar value by definition. In particular, spectral flux calculates a scalar value that represents the sum of the spectral difference between two different audio spectra. So, to implement this, we chose to compare successive pairs of Short-time Fourier Transform (STFT) frames. As this calculation results in a vector (array), we opted to calculate the variance, which returns a value corresponding to how far the data is spread out.

Within our API, students can choose to analyse segments of audio files (or entire files) or segments of tracks (or entire tracks). Figure 6 shows a brief example: RMS amplitude is calculated for every semiquaver of an audio file and the segments are then sorted from lowest to highest RMS value.

To implement these analysis routines, we initially considered leveraging the Echo Nest Remix Python API, which calculates low-level features as well as information such as tempo, key and 'dance-ability' (The Echo Nest 2012). However, this approach had several drawbacks. Since computation is done in Echo Nest's cloud, it introduces a significant delay while uploading audio content to the remote server, the service limits the total number of requests per user per hour, and the service is unavailable entirely to those without Internet access (or with regulated Internet access, as at most secondary schools). So, instead, we took advantage of ReaScript's ability to return audio sample data as Python lists and implemented the analysis routines locally using NumPy and SciPy (SciPy 2012, NumPy 2012).

4.1.3. Synthesis and effects creation

Holistic introductory computer music courses also typically teach the basics of digital signal processing and sound synthesis. While existing software synthesisers and effects units are a great aid to pedagogy in this regard, it is also important for students to design their own synthesisers and signal processors. This is typically accomplished using a unit-generator architecture such as Max/MSP, SuperCollider or ChucK, since the unit-generator approach has been the dominant one in computer music since MUSIC III and is itself inspired by the paradigm established by modular analogue synthesis (Wang 2007).

To this end, we are in the process of developing an architecture where students can create synthesis instruments and signal processing effects that can be inserted onto tracks within Reaper, by writing similar Python code within their EarSketch script. This capability thus brings composition, signal processing, feature extraction and sound design into the same environment and gives students the chance to immediately use the plugins they program inside of commercial DAW software.

Our architecture will enable students to create and connect unit-generators and specify control parameters that can be accessed through current API functions (i.e. through setEffect()). When students run EarSketch scripts that use the unit-generator architecture to define new effects, EarSketch automatically translates the effects units into JesusSonic (JS) code, an interpreted language native to Reaper for defining new effects. We have already developed a proof-of-concept implementation in which we successfully created and connected unit-generator oscillators with EarSketch and then created effects automations for them with the setEffect() function. We are now developing a more fully featured, robust implementation.

4.2. Social media website

Our social media web site (see Figure 7) facilitates informal sharing and borrowing of code among students. Following the paradigm established by online music remixing communities such as ccMixter (ccMixter 2012) and computing education communities such as Scratch (Scratch 2012), the site enables students to browse projects created by their peers, listen to the audio and view the code inside the browser, download the project, add comments and tags, and see how code has been reused and transformed by others. To facilitate seamless integration, we created a menu command within Reaper that automatically uploads a user's relevant source code and audio files for a project to his or her social media site account.

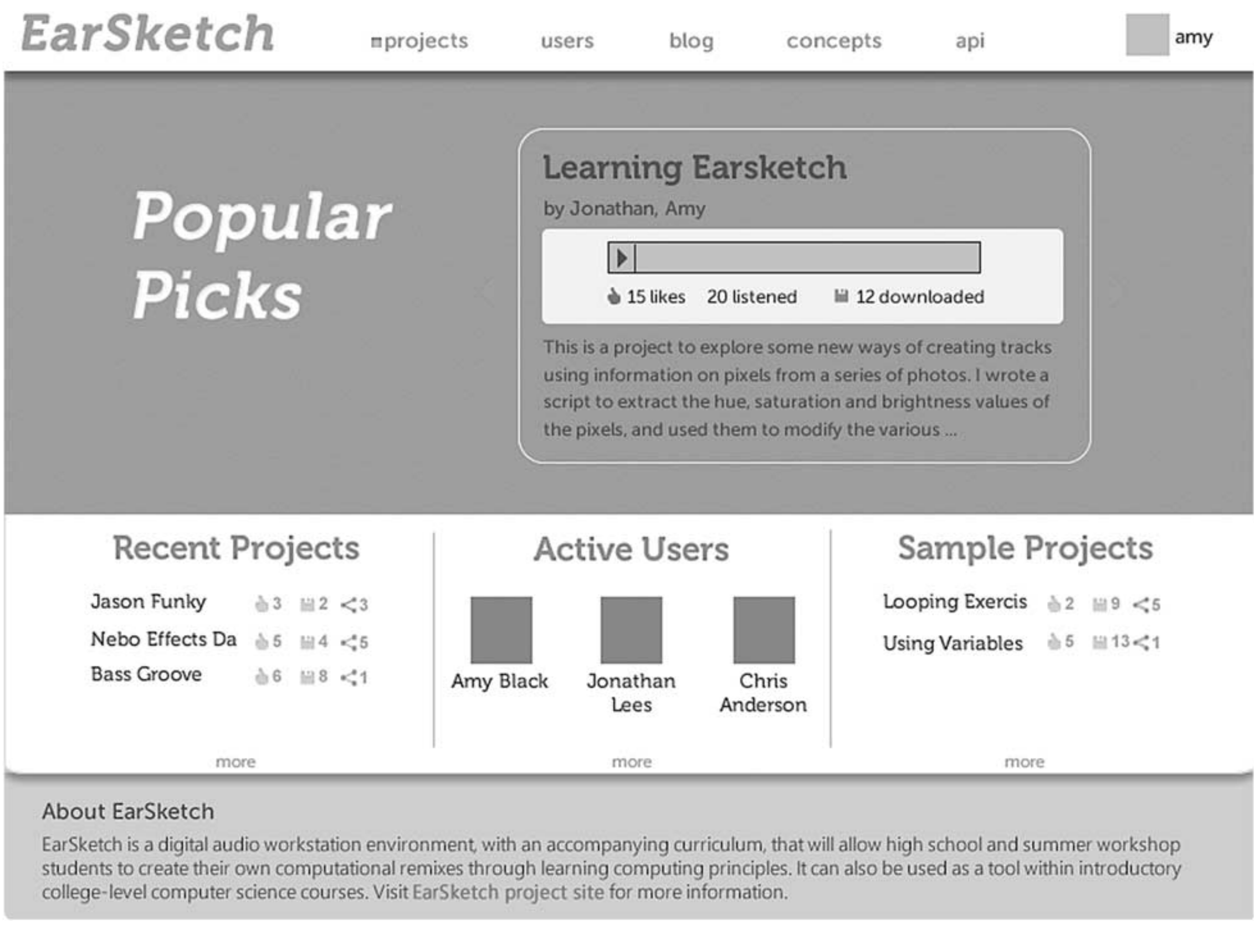
5. SUMMER WORKSHOP PILOT

In July 2012, we conducted the first EarSketch workshop to pilot the project. In accordance with the primary interest of EarSketch's funding agency, the workshop focused on computer science education as motivated by a connection to computer music. It also introduced basic computer music concepts such as multi-track digital audio sequencing, effects and automations, and step sequencing, and compositional concepts such as ABA and verse–chorus–bridge form and stochastic techniques.

The workshop consisted of 5 days of instruction, approximately 5 hours per day, with each day being divided into 60-minute segments. These segments were designed to provide a comfortable amount of time for instruction and demonstration with longer periods of time for individual student practice and application, followed by sharing and reflection. Students had specific assigned exercises, more open-ended work time, and round-robin and pair-programming activities. Besides the instructor lessons, they also had access to an EarSketch curriculum manual and reference documentation. Ultimately, the students worked to create a final project to present and share with family members during an 'open house', which took place on the final afternoon of the workshop.

5.1. Curriculum design

Given the pedagogical focus of the summer workshop on computer science, the curriculum took a combined pedagogy approach (see section 2.3). It incorporated basic computing concepts (common to most all introductory computing courses) grounded in 'thick authenticity', which focuses on making learning experiences personally meaningful and relevant to the larger world (Shaffer and Resnick 1999). In this case, of course, computer music composition is the context that is personally meaningful for students,



Learning Earsketch

by Jonathan, Amy

15 likes 20 listened 12 downloaded

This is a project to explore some new ways of creating tracks using information on pixels from a series of photos. I wrote a script to extract the hue, saturation and brightness values of the pixels, and used them to modify the various ...

Jason Funky	3	2	3
Nebo Effects Da	5	4	5
Bass Groove	6	8	1

Amy Black
Jonathan Lees
Chris Anderson

Looping Exercis	2	9	5
Using Variables	5	13	1

About EarSketch

EarSketch is a digital audio workstation environment, with an accompanying curriculum, that will allow high school and summer workshop students to create their own computational remixes through learning computing principles. It can also be used as a tool within introductory college-level computer science courses. Visit EarSketch project site for more information.

Figure 7. Homepage of the social media website.

as they apply computing concepts towards creating their own music.

5.1.1. Overall design

The EarSketch curriculum presumes no prerequisites: it does not expect students to play a musical instrument or read music notation. It also assumes that students have no prior experience in using music software or in programming. Its design emphasises hands-on, collaborative experiences.

The curriculum also emphasises some of the unique benefits of approaching music from a computational perspective. In particular, it demonstrates to students how to rapidly experiment with many possible variations on a musical idea by iteratively modifying algorithmic parameters, executing each version of code and using their musical ears to decide which option(s) are best (Ericson, Guzdial and Biggers 2007). For example, they may explore how many times a loop should be repeated before a new one is introduced, or try several analysis features before determining which to use within an algorithm. The curriculum also focuses on the classic idea of a composer as a ‘pilot’ who uses computation to negotiate low-level details so as to focus more on higher-level musical structures and on the ways in which a computational interface transforms musical thinking (Xenakis 1992).

5.1.2. Digital Audio Workstation

The first topic covered in the curriculum is the Reaper DAW. Before coding, students first learn how to create music using Reaper’s standard graphical interface. This familiarises them with music industry software paradigms, helps them to build a mental model of the DAW as a foundation for further programming abstractions, provides clear connections between API functions and standard DAW functionality (e.g. insertMedia() corresponds to importing an audio clip onto the timeline), and demonstrates the unique advantages and time savings programming can offer in executing complex tasks, such as inserting effects automations at each semiquaver location on a particular track.

5.1.3. Programming

Programming concepts in the EarSketch curriculum are introduced hand in hand with DAW capabilities and musical constructs. The current workshop curriculum covers basic Python syntax (e.g. comments, imports); the use and declaration of variables, functions and parameters; iteration with for loops, index variables and skip counting; strings; simple maths and random functions; conditionals; and list data structures. The curriculum always motivates use of these computing concepts by linking them to new methods of musical composition in the DAW. For example, students declare functions to generate A and B sections of music and then use the flexibility that such abstractions enable to quickly modify, move and repeat these musical passages. They use loops with skip-counting indices to create musical variation at structurally significant time intervals in the music. They create strings to define their own step-sequenced rhythmic patterns within the music. They use random functions to select among different audio files or rhythmic patterns stored in a list to create unexpected musical results that change each time a program is executed.

The curriculum also emphasises the ways in which programming can simplify the execution of complex tasks that would be tedious to do in Reaper's graphical interface, thus permitting rapid, iterative musical experimentation. For example, students first learn how to apply effects by using Reaper's GUI to draw automation envelopes. They then use the setEffect() function to more easily create and control effects and parameter changes programmatically.

Collaboration and sharing are incorporated into the curriculum at all levels. After each lesson, students have time to work on their own projects, frequently collaboratively. We use 'round robin' and 'pair' style coding, where students work for five minutes on a project, then continually shift one seat over in the classroom to a new project or alternate back and forth with a single collaborator on the same project. All of this work is placed onto the social media website, allowing students to view and download other students' projects. By appropriating musical ideas and programming techniques from uploaded work, collaborative practices emerge even without direct pairing or contact.

5.2. Evaluation

The seventeen high school students that participated in the EarSketch workshop were from a variety of Metro Atlanta schools. The ethnic demographic of the group was relatively diverse, with 18 per cent of the students being Asian-American, 24 per cent being African-American, 53 per cent European-American, and 6 per cent of two or more ethnicities. The class comprised 75 per cent male and 25 per cent female students.

Programme participants were measured at three time points over their 5-day EarSketch experience: a ten-item pre-content knowledge assessment on day 1, an engagement survey at the end of day 4, and the same content knowledge assessment on day 5 that the students saw on day 1. The evaluation instruments focused more on computing than on computer music, again because of the specific interests of the project's funding agency.

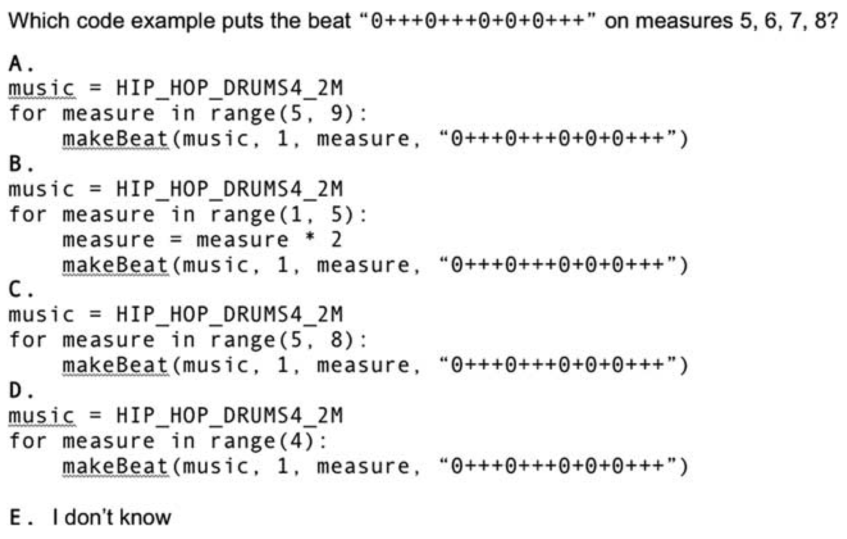
The content knowledge assessment is designed to measure basic knowledge related to programming within a DAW environment. Figure 8 shows an example of a question.

On day 4, participants were asked to complete a survey measuring seven engagement constructs (Wiebe, Williams, Yang and Miller 2003). The engagement instrument comprises five scales from Williams, Wiebe, Yang, Ferzli and Miller (2002):

- computing confidence (e.g. 'I am sure I can do advanced work in computing');
- computing enjoyment (e.g. 'I feel comfortable working with a computer');
- perceived usefulness of computing (e.g. 'I will be able to get a good job if I learn how to use a computer');
- motivation to succeed in computing (e.g. 'I like solving computing problems'); and
- computing identify and belongingness (e.g. 'I feel like I "belong" in computer science').

The instrument also draws on a scale designed to measure the extent to which participants see computing as a creative outlet (e.g. 'I am able to be very expressive and creative while doing computing') (Knezek and Christensen 1996). Finally, the instrument measures students' intention to persist in computing as a field of university study and a career (e.g. 'Someday, I would like to have a career in computing'). The student engagement survey was administered retrospectively, such that students' answered how they felt before the week-long EarSketch camp and at the end. The analysis consisted of a paired samples t-test whereby statistically significant changes from before to after were assessed.

Overall, the results suggest that students' attitudes positively and statistically significantly increased across three constructs at $p < .01$: computing confidence, motivation to succeed in computing, and creativity. This suggests that the workshop is effective at enhancing students' perceptions that they can tackle advanced computing work and get good grades in computing (computing confidence). Likewise, the workshop increases students' motivations to persevere with complex computing problems (motivation to succeed in computing). Originality and expressivity are also enhanced as a result (creativity). Students indicate that they are statistically significantly more likely to see themselves in a computing field in the future as a result of the workshop.



Which code example puts the beat “0+++0+++0+0+0+++” on measures 5, 6, 7, 8?

```
A.
music = HIP_HOP_DRUMS4_2M
for measure in range(5, 9):
    makeBeat(music, 1, measure, “0+++0+++0+0+0+++”)
B.
music = HIP_HOP_DRUMS4_2M
for measure in range(1, 5):
    measure = measure * 2
    makeBeat(music, 1, measure, “0+++0+++0+0+0+++”)
C.
music = HIP_HOP_DRUMS4_2M
for measure in range(5, 8):
    makeBeat(music, 1, measure, “0+++0+++0+0+0+++”)
D.
music = HIP_HOP_DRUMS4_2M
for measure in range(4):
    makeBeat(music, 1, measure, “0+++0+++0+0+0+++”)
```

E. I don't know

Figure 8. Example of a moderate content knowledge assessment question.

Table 1. Change from before to after along the seven engagement constructs

	Constructs								
	Paired samples								
Constructs		**n**	**Mean**	***t*-test**	**SD**	**D**	**N**	**A**	**SA**
Computing Confidence	Before	16	3.20	0.008**	14%	18%	24%	22%	21%
	Now	17	4.11		3%	3%	18%	33%	43%
Computing Enjoyment	Before	17	4.31	0.838	2%	5%	11%	25%	57%
	Now	17	4.34		1%	5%	10%	27%	57%
Perceived Usefulness of Computing	Before	17	4.39	0.945	2%	1%	9%	32%	56%
	Now	17	4.38		3%	6%	8%	17%	66%
Motivation to Succeed in Computing	Before	17	3.57	0.000**	3%	19%	21%	34%	24%
	Now	17	4.03		1%	7%	17%	39%	36%
Computing Identity and Belongingness	Before	17	3.59	0.277	2%	20%	27%	20%	31%
	Now	17	3.76		4%	14%	22%	24%	37%
Intention to Persist	Before	17	3.58	0.245	5%	12%	33%	22%	28%
	Now	17	3.70		5%	10%	28%	26%	32%
Creativity	Before	17	3.46	0.006**	9%	5%	38%	28%	21%
	Now	16	4.20		0%	0%	19%	42%	39%

**p < .01; *p < .0l; negatively worded items were reverse coded prior to mean computation of the constructs. Only students with matched before and now scores were assessed for statistical significance. Note: We employed a 5pt. scale with the following response categories: Strongly Disagree (SD), Disagree (D), Neutral (N), Agree (A), and Strongly Agree (SA).

To further investigate how the EarSketch programme impacted students' intent to persist in computing, a correlation analysis was conducted whereby the change in responses (Δ) from ‘before’ to ‘now’ across the seven survey constructs listed in Table 1 were entered into a Pearson's correlation. The results, presented in Table 2, indicate that to the extent that students gained in confidence, belongingness and creativity, their intent to persist in computing increased. Interestingly, growth in creativity was statistically significantly correlated with growth in confidence, motivation and belongingness. This suggests that the computer music context for computing education may have a powerful, positive effect on student confidence and motivation.

Figure 9 summarises students' gains in content knowledge across all content knowledge assessment items, specifically examining content knowledge gains by item difficulty. The results clearly suggest that the workshop was statistically significantly effective in enhancing students' knowledge of programming concepts. On average, students made a four-item gain from before (pre) to after (post); that is, the average student answered six questions correctly after the sessions and only two questions before.

Table 2. Pearson correlation

	Δ CKA overall	Δ Intent to Persist	Δ Confidence	Δ Enjoyment	Δ Importance & Perceived Usefulness	Δ Motivation	Δ Identity & Belongingness	Δ Creativity
Δ CKA (overall)	–							
Δ Intent to Persist	.083	–						
Δ Confidence	−.175	.583*	–					
Δ Enjoyment	.282	.093	.540*	–				
Δ Importance & Perceived Usefulness	.365	.366	.628**	.912**	–			
Δ Motivation	−.149	.431	.574*	.408	.364	–		
Δ Identity & Belongingness	.139	.740**	.815**	.646**	.839**	.507*	–	
Δ Creativity	−.242	.780**	.775**	.191	.333	.611**	.691**	–

**p < .01; *p < .05. Δ = Now-Before

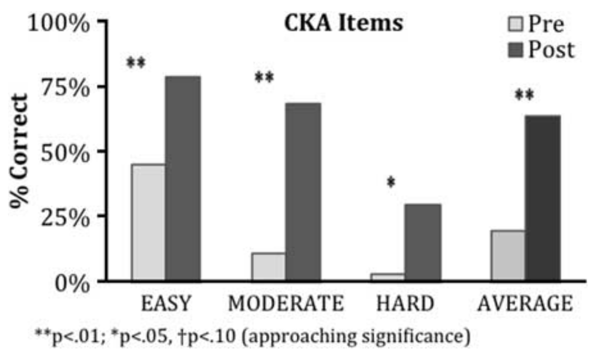


Figure 9. Pre/post content knowledge assessment items by difficulty.

When asked what aspects of the workshop students enjoyed the most, the vast majority indicated that they appreciated learning about programming and working on complex and interesting projects in computing and music. For example, one student noted that 'the best thing about this camp was the fact that we were introduced to programming, while also integrating it into musical concepts'. Another student expressed her enthusiasm for the workshop projects by saying, 'Learning how to create music using only code with minimal help from a DAW was very exciting and interesting. The code allows for a much easier and faster way to mix and remix music.' This response – that coding can actually make it easier to create music than more traditional music production techniques – exactly matches one of our intents in creating EarSketch. Through an integrated software environment, the students were able to easily see the unique benefits of programmatic and algorithmic techniques.

6. FUTURE WORK

We continue to iteratively review, refine and evaluate EarSketch as we prepare for a public release and broader deployment of the project. Near-term plans include a six-week pilot at an ethnically and socioeconomically diverse public high school in metro Atlanta in early 2013, to assess the success of our workshop curriculum in the context of a high school introductory computer science course; and a two-week summer workshop in 2013 at Georgia Tech's Institute for Computing Education, to pilot additional curriculum modules focusing on more advanced computing, composition and computer music concepts.

We are also adding new functionality to the EarSketch software toolset and social media website. Some of our development is focused on improving usability and the seamless integration of multiple software components through changes such as new EarSketch menu items within Reaper and a cleaner project metadata entry page on the social media website. Other changes focus on new API functionality for the software toolset, in particular on completing the implementation of the unit-generator-based synthesis and signal processing components of the API.

We are currently replacing the existing library of audio content bundled with EarSketch, which was licensed to us only for use in pilot studies, with a more openly licensed library of audio content. Specifically, we commissioned two sound designers – Young Guru and Richard Devine – to create new audio library content for EarSketch in a variety of musical genres.

Lastly, we are planning to use EarSketch in a massively open online course (MOOC) in Music Technology through Coursera (Coursera 2012). Because of the interests of our funding agency, our pilot studies to date have emphasised computer science over electroacoustic music composition in their curricula. With the MOOC platform, we are now able to offer a course that weights computing and composition more equally, and we can offer it for free to a student population in the tens of thousands. This will fulfil one of the core motivating visions of the project: to bring together a holistic introductory computer music curriculum, an integrated, affordable software environment useable on any laptop, and an online community through which to share music and code and create derivative works, in an effort to expand access to, and interest in, the artistic pursuit and research practice of computer music.

Acknowledgements

Our thanks to everyone that has contributed to and collaborated on this project: Barbara Ericson, Ziwen Fan, Charles Hancock, Tom Jordan, Elise Livingston, Donovan McMurray, Justin Permar, Mike Reilly, Erica Richards, Jonathan Streater, Neeraj Vaidya and Shannon Yao. Additional thanks to Cockos, Inc., to Georgia Tech's Institute for Computing Education, to Nebo Elementary School, to the Marist School and to Lanier High School for their guidance, collaboration and support. This work is funded by the National Science Foundation under CNS grant #1138469. In spring 2013, EarSketch will be available as a free download at http://earsketch.gatech.edu.

REFERENCES

Ballora, M. and Craig, C. 2010. Studio Report: Music Technology At Penn State University. *Proceedings of the 2010 International Computer Music Conference*. New York: ICMA, 286–9.

Burk, P., Polansky, L., Repetto, D., Roberts, M. and Rockmore, D. 2011. Music and Computers: A Theoretical and Historical Approach. http://music.columbia.edu/cmc/MusicAndComputers.

Burtner, M. 2012. Technosonics: Digital Music and Sound Art Composition. http://people.virginia.edu/~cmb4f/235/235.html#syllabus.

ccMixter. 2012. http://ccmixter.org.

Cipriani, A. and Giri, M. 2011. Innovation, Interaction, Experience and Imagination in Computer Music Education. *Proceedings of the 2011 International Computer Music Conference*. University of Huddersfield: ICMA, 383–6.

Collins, N. 2010. *Introduction to Computer Music*. West Sussex: John Wiley & Sons.

Coursera. 2012. https://www.coursera.org.

Davis, P. 2011. http://jackaudio.org.

Didkovsky, N. and Burk, P. 2004. Java Music Specification Language. http://algomusic.com/jmsl.

Dodge, C. and Jerse, T.A. 1997. *Computer Music*, 2nd edn. New York: Schirmer Books.

Echo Nest, The. 2012. http://echonest.github.io/remix.

Ericson, B., Guzdial, M. and Biggers, M. 2007. Improving Secondary CS Education: Progress and Problems. *ACM SIGCSE Bulletin*, 298–301.

Essl, G. 2010. The Mobile Phone Ensemble As Classroom. *Proceedings of the 2010 International Computer Music Conference*. New York: ICMA, 506–9.

Eugene, W. and Gilbert, J.E. 2008. C-PAL: Culture-based Programming For Adult Learners. *Proceedings of the 46th Annual ACM Southeast Regional Conference*, ACM, 450–3.

Garton, B. and Diels, N. 2012. MIDI Music Production Techniques. http://music.columbia.edu/cmc/courses/v2205/fall2012/syl.html.

Grosse, D. 2008. Loop File Formats. http://www.recordingmag.com/resources/resourceDetail/323.html.

Guzdial, M. 2003. A Media Computation Course For Non-Majors. *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*. New York: ACM, 104–8.

Hamer, J. 2004. An Approach to Teaching Design Patterns Using Musical Composition. *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. Leeds: ACM, 156–60.

Kelleher, C. and Pausch, R. 2005. Lowering the Barrier to Programming: A Taxonomy of Programming Environments and Languages For Novice Programmers. *ACM Computer Survey* **37**(2): 83–37.

Kelleher, C., Pausch, R. and Kiesler, S. 2007. Storytelling Alice Motivates Middle School Girls to Learn Computer Programming. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. San Jose, CA: ACM, 1455–64.

Klingbeil, M. 2009. Music 325a: Fundamentals of Music, Multimedia Art, and Technology. http://musi325_fall09.commons.yale.edu.

Knezek, G. and Christensen, R. 1996. Validating the Computer Attitude Questionnaire (CAQ). *Paper Presented at the Annual Meeting of the Southwest Educational Research Association*. New Orleans, LA.

Magnusson, T. 2011. The IXI Lang: A SuperCollider Parasite for Live Coding. *Proceedings of the 2011 International Computer Music Conference*. University of Huddersfield: ICMA, 503–6.

Malan, D. and Leitner, H. 2007. Scratch For Budding Computer Scientists. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. Covington, KY: ACM, 223–7.

MEAPsoft. 2008. http://www.meapsoft.org.

MIDI Manufacturers Association. 2012. http://www.midi.org.

Misra, A., Blank, D. and Kumar, D. 2009. A Music Context For Teaching Introductory Computing. *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science*. Chattanooga, TN: ACM, 248–52.

Monroy-Hernández, A. and Resnick, M. 2008. Empowering Kids to Create and Share Programmable Media. *Interactions* **15**(2): 50–3.

Moore, F.R. 1990. *Elements of Computer Music*. Englewood Cliffs, NJ: Prentice Hall.

NASM (National Association of Schools of Music). 2012. *National Association of Schools of Music Handbook 2011–12*. http://nasm.arts-accredit.org/site/docs/Handbook/NASM_HANDBOOK_2011-12.pdf.

Nielson, S.J. and Knutson, C.D. 2004. OO++: Exploring the Muliparadigm Shift. *Proceedings of the Workshop on Multiparadigm Programming with Object-Oriented Languages*. Oslo.

NumPy. 2012. http://www.numpy.org.

Pope, S.T. 1994. Editor's Notes: A Taxonomy of Computer Music. *Computer Music Journal* **18**(1): 5–7.

Preston, J. and Morrison, B. 2009. Entertaining Education-Using Games-Based and Service-Oriented Learning to Improve STEM Education. *Transactions on Edutainment III*:70–81.

Propellerhead Software AB. 2012. http://www.propellerheads.se/developer/index.cfm?fuseaction=get_article&article=rewiretechinfo.

Rich, L., Perry, H. and Guzdial, M. 2004. A CS1 Course Designed to Address Interests of Women. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*. Norfolk, VA: ACM, 190–4.

Roads, C. 1996. *The Computer Music Tutorial*. Cambridge, MA: The MIT Press.

Ruthmann, A., Heines, J.M., Greher, G.R., Laidler, P. and Saulters, C. 2010. Teaching Computational Thinking Through Musical Live Coding in Scratch. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. Raleigh, NC: ACM, 351–5.

Schedel, M. 2007. Electronic Music and the Studio. In N. Collins and J. d'Escriván (eds.) *The Cambridge Companion to Electronic Music*. New York: Cambridge University Press.

SciPy. 2012. http://www.scipy.org.

Scratch. 2012. http://scratch.mit.edu.

Shaffer, D. and Resnick, M. 1999. 'Thick' Authenticity: New Media and Authentic Learning. *Journal of Interactive Learning Research* **10**(2): 195–215.

Shannon, C. 2003. Another Breadth-First Approach to CS 1 Using Python. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*. Reno, NV: ACM, 248–51.

Sloan, R.H. and Troy, P. 2008. CS 0.5: A Better Approach to Introductory Computer Science For Majors. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. Portland, OR: ACM, 271–5.

TIOBE Software. 2012. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

Wang, G. 2007. A History of Programming and Music. In N. Collins and J. d'Escriván (eds.) *The Cambridge Companion to Electronic Music*. New York: Cambridge University Press.

Wang, G., Trueman, D., Smallwood, S. and Cook, P.R. 2008. The Laptop Orchestra as Classroom. *Computer Music Journal* **32**(1): 26–37.

Webster, P. and Williams, D. 2006. *Experiencing Music Technology*, 3rd edn. Belmont, CA: Thomson, Schirmer.

Wiebe, E., Williams, L., Yang, K. and Miller, C. 2003. Computer Science Attitude Survey. *Computer* **14**(25): 1–86.

Williams, L., Wiebe, E., Yang, K., Ferzli, M. and Miller, C. 2002. In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education* **12**(3): 197–212.

Xenakis, I. 1992. *Formalized Music: Thought and Mathematics In Composition*. Hillsdale, NY: Pendragon Press.