Collaborative Coding with Music: Two Case Studies with EarSketch

Avneesh Sarwate
Georgia Tech Center for Music Technology
Atlanta, Georgia
avneesh@gatech.edu

Takahiko Tsuchiya
Georgia Tech Center for Music Technology
Atlanta, Georgia
takahikio@gatech.edu

Jason Freeman
Georgia Tech Center for Music Technology
Atlanta, Georgia
jason.freeman@gatech.edu

ABSTRACT

This paper describes the motivation, design, and implementation of new features in EarSketch that enable the collaborative creation of algorithmic music. EarSketch is a web-based Digital Audio Workstation (DAW), designed primarily for educational contexts, in which users author Python or JavaScript code to programmatically create music within a multi-track paradigm. In this paper, we describe these new collaborative features in EarSketch and discuss their potential for use in both educational and music performance contexts.

1. INTRODUCTION

By integrating features to support multi-user interaction across a range of collaborative paradigms, systems for musical coding can facilitate new types of creativity, expression, and communication in both educational and performance contexts [2]. While tools for collaborative coding — which range from version control systems such as git and SVN to real-time systems such as FirePad and CodeBunk — collaborative coding systems designed specifically for music are still an emerging area of research.

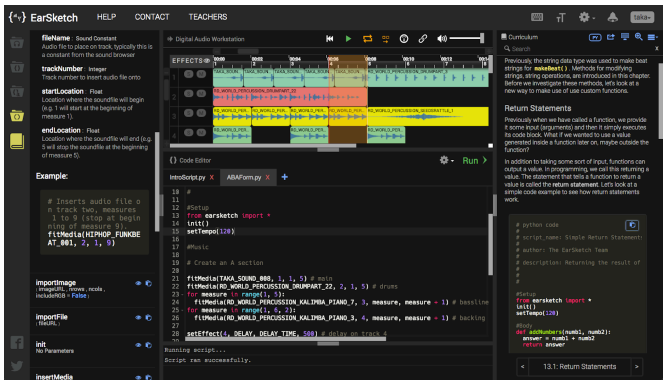

Figure 1: The EarSketch Interface

EarSketch (Figure 1) engages a diverse population of students in introductory computer science by teaching coding in the context of algorithmic composition [9]. It consists of a web-based programming environment, digital audio workstation, curriculum, and audio loop library that enables students to write Python or JavaScript code to algorithmically create music in popular genres. EarSketch students write code to creatively manipulate musical samples while learning computing fundamentals such as loops, lists, and functions. The platform, currently implemented as a web-based application using Web Audio API, has been used by over 220,000 users to date.

In recent years, students and teachers have increasingly requested new features in the platform to support collaborating on EarSketch projects and sharing those projects. Since the launch of the new AP Computer Science Principles course in the United States in 2016 [1] — a course to which the EarSketch curriculum is closely aligned — collaboration has become increasingly important as one of six computational thinking practices highlighted in the course. The EarSketch curriculum includes modules which introduce students to key concepts in collaboration and peer critique, such as Liz Lerman's Critical Response Process [8], then ask them to collaborate on a project, and finally challenge them to reflect on how they collaborated.

This paper focuses specifically on the new collaboration features in the EarSketch web-based platform itself that we have developed to support the growing focus on collaboration and sharing that students, teachers, and new curricular frameworks have demanded. After reviewing core concepts in computer-supported cooperative work, collaborative coding, and live coding, this paper focuses on the design and implementation of the new collaboration features in EarSketch. It also explains the applications of these collaborative paradigms not only in educational settings but also in ensemble live-coding performance contexts (such as laptop orchestras). Technical challenges and solutions in the implementation of these collaborative features are also reviewed in the broader context of real-time audio systems using Web Audio API.

1.1 Computer Supported Cooperative Work (CSCW)

Collaborative coding in the creative world has taken on many flavors, and a useful framework for discussing collaborative music systems generally is presented by Barbosa [2]. Barbosa's framework, which closely follows more general ones established in the CSCW community, defines collaborative contexts across two axes: location (collaborators can be either co-located or remote), and time synchronization (e.g. "synchronous" - actively working together in real time, or "asynchronous" - sharing artifacts and working at different times). A system such as Splice (a cloud based tool for sharing DAW project versions) would be an example of a remote/asynchronous collaboration system, while an instance of co-located/synchronous musical collaboration would be a duo performing together on the same set of DJ decks.

1.2 Collaborative Coding

There currently is a diverse ecosystem of collaborative coding tools, both musical and otherwise. As mentioned above, version control systems such as Git or SVN could be considered the most popular collaborative coding tools, allowing up to thousands of people to contribute code to the same project. Another format of popular collaborative coding tools is the "notebook", as popularized by Mathematica and the Jupyter project [6]. The sharing of notebooks allows for a potentially faster (though still non-realtime) form of collaboration by sharing smaller, more self-contained chunks of code whose results are displayed inline with the code itself. There are also a variety of real-time collaborative coding tools - editors such as Firepad and CodeBunk that allow remote collaborators to write and compile code in real-time right in the browser.

Real-time code collaboration is an approach also adopted by many musical tools. In fact, there are many tools that support the practice of "live coding", that is, creating live music and/or graphics by writing code in real-time in front of an audience [3]. LOLC (Figure 2), a combination chat and coding environment [7], lets several co-located performers on the same LAN perform music by sharing snippets of code into a communal chat room.

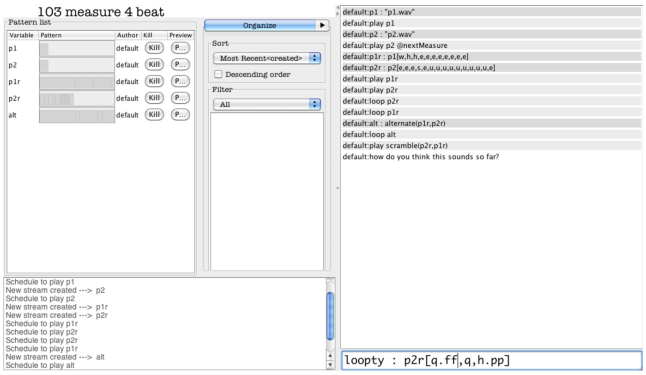

Figure 2: The LOLC interface

Another LAN based collaborative system is Troop [5]. Troop is a collaborative text editor where only one of the user is a "server" who renders the final audio. Troop supports the musical live-coding languages TidalCycles (Haskell) [12], FoxDot (Python) [4], and SuperCollider [10], and can be configured to work with any language that supports REPL style execution. Thus, users can collaboratively build a document in Troop, but only execute certain lines at a time. Estuary (Figure 3), a web based environment, allows remote performers to collaborate on a musical performance by writing code in TidalCycles, a Haskell based music library [14]. In Estuary, each performer is given their own editing window, but performers can break convention and edit the code in another performer's window. All audio from each space is rendered locally for each performer, and not necessarily synchronized between performers. Gibber is another live-coding environment that allows for collaborative editing [15], implementing a similar "one space per user" pattern to Estuary.

Web Audio Conference WAC-2018, September 19–21, 2018, Berlin, Germany.

Figure 3: The Estuary interface

2. PRIOR WORK WITH EARSKETCH

2.1 Sharing

Earlier versions of EarSketch supported user project sharing but not user collaboration. Users could share projects with others via a permalink URL, directly with other EarSketch users, or via SoundCloud (in which case the source code would be posted in the track's description field). When a shared project was opened in EarSketch, it was read-only: users could see the code and the multi-track DAW render, but the only way they could edit the script was to create a duplicate copy as a new project.

This sharing functionality supported two primary purposes. First, students shared projects with their teacher so they could be graded, typically by pasting a permalink URL into a learning management system (like Google Classroom or Canvas) or by sharing the project directly with the teacher's EarSketch account. Second, students shared projects with friends and family, typically via the permalink, SoundCloud, or by simply exporting their project as an MP3. (Our research has shown a close correlation between students' interest in sharing their projects and their intention to persist in computing, as reported in [11]).

These sharing tools, however, did little to support student collaboration, because sharing was always unidirectional: one user could share a project with another user, but there was no way for edits to sync back to the original author's account. For example, we often observed a pair of students working side by side together on an EarSketch project, using two adjacent computers in a lab. They would delegate different functions for each student to write and then work separately on the two computers. When they tried to integrate their work, they inevitably ran into trouble. One student would share their script with the other student, who would then perform the integration work. But the first student had no way to access or edit the combined script unless it was shared back to them. Inevitably, versions would get out of sync and students would be unable to find the right code from one day to the next.

2.2 Live Coding

EarSketch was originally designed for batch-style execution: users would write their code and then press a "run" button to render the code in visual (DAW tracks) and audio form. EarSketch was later extended to support a live coding model as well [18]: the code could be re-run during playback and the DAW tracks and audio would update with minimal interruption to playback. But this live-coding mode was not in any way collaborative: it was useful for teachers modeling coding practices in front of a class and for students experimenting with iterative modifications to their code and music, but lacked features such as time synchronization to make it useful in a group performance context.

In the following sections we outline two use cases for new collaboration features in EarSketch — group class projects and large-ensemble live-coding performance — and discuss the design,

implementation, and implications of the features to support these settings in detail.

3. USE CASE #1: GROUP PROJECTS IN CLASSES

3.1 Context

To support collaborative work on student projects, we added features to EarSketch to support multiple boxes of the CSCW matrix and to move beyond unidirectional, read-only script sharing. In addition to script sharing, users can now grant edit access to other EarSketch users, and those scripts can then be edited synchronously or asynchronously by any user with edit access. The real-time collaborative text editor includes standard features, such as highlighting which users are currently editing the script and showing the cursor position of each active user.

We encourage the use of this feature in classrooms in the form of a "jukebox challenge." In this activity, a small group of students are asked to create short algorithmic pieces and later integrate them into a jukebox script that accepts a user input or uses random value for song selection. The integration process involves the peer reviewing of songs, adapting each other's functions, and resolving potential name conflicts. Shared editing eases the merging of scripts in this collaborative process, and real-time visual feedback in the script editor facilitates greater interactivity in synchronous peer-reviewing processes.

3.2 Design and Implementation

The new script-sharing mode, "Let Others Edit", enables synchronous and asynchronous Google-Docs-style collaborative editing of an EarSketch script.

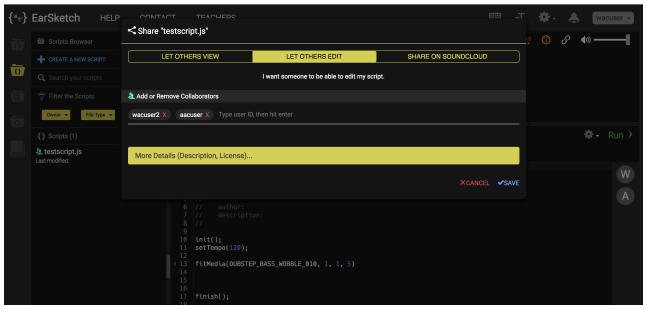

Figure 4: The collaboration invitation interface

By inviting a user, it turns a regular script (tab) to a shared workspace with multiple authors. Like Google Docs, this mode employs the operational-transform (OT) algorithm [17] via websockets data exchange. OT, in essence, synchronizes multiple clients' local text edits to the server version by recording and adjusting individual edit operations when they are transmitted out of order (i.e., concurrently) by the clients.

This generic feature was an essential baseline technology to facilitate synchronous peer learning and exploration with continuity and granularity, enabling all four of the collaborative configurations discussed above (i.e., the cross section of co-located vs remote, and synchronous vs asynchronous). However, it has also highlighted new design challenges. For example, there is a need for a new overlay UI that would enable a teacher or a team "navigator" to moderate and guide the student learning and creative work. Our previous study examined generic communication tools (e.g., chat room and hashtag-typed messaging) as well as more integrated UI solutions such as a turn-taking UI (though it is a challenge to create a fluid workflow with this), inline indicators for the new user edits (while keeping the syntax highlighting), and a code-block and snippet management UI [16].

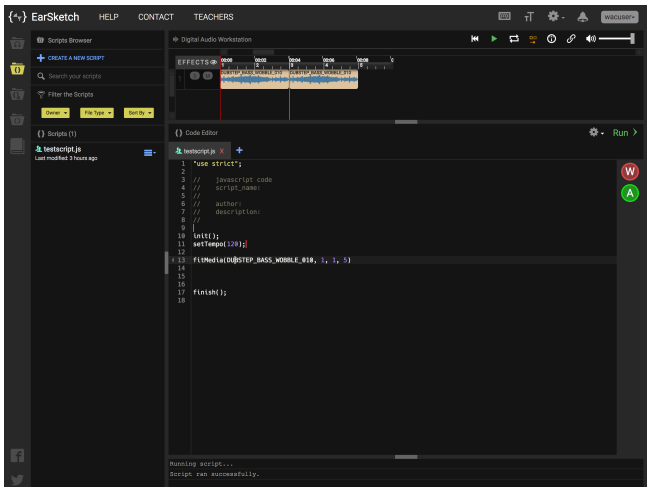

Figure 5: The EarSketch Shared Editing interface

3.3 Applications

3.3.1 Co-located/Synchronous

In a classroom setting, where students are co-located and working together in real-time, EarSketch can serve as a tool for making lecture style lessons more interactive. During a lesson where a teacher is presenting a programming concept with code examples, rather than showing coding on slides or on a non-interactive script, the teacher could share a single script with the entire class in a single collaboration session, as well as projecting it onto the screen. This would allow for interactivity in several ways. The instructor could ask students to implement small tasks or functions (analogous to calling a student “up to the chalkboard” to solve problems). This would allow instructors to organically show different problem solving strategies and pitfalls as they arise [18]. Also, students could leave comments in the code where they have questions, allowing the teacher to note problem points in the lesson without necessarily disrupting it. On a more mundane level, screen sharing can help students to see the code a teacher is typing by mirroring it on their own computer screens: in many secondary schools, computer labs have awkward physical layouts and projection screens are quite small, making it hard for many students to see a projected screen even with enlarged type.

The live-collaboration feature can also ease the process of pair-programming in the classroom. Providing each student with a real-time collaborative editor can enable students to swap between “navigator” and “driver” roles more fluidly, especially if there are more than two students working together.

3.3.2 Remote/Synchronous

The collaborative editor also allows students to pair or group program remotely. When combined with an audio or video chat, the ability to swap between “driver” (the person coding) and “navigator” (the person watching, making suggestions and corrections) and speak with collaborators in real-time can replicate many facets of the in person experience. The ability to collaborate in real-time outside the classroom greatly expands the potential for peer-learning and development of collaboration skills.

The collaborative editor can also be used to enable remote teaching assistant (TA) sessions. Similar to the classroom use-case, an instructor could either present or answer questions about a piece of

code in an interactive setting in real time, with students joining the TA session both by accessing the script and entering a conference call.

3.3.3 Remote/Asynchronous

For collaborative scripts, any user can edit the script at any time. This can allow students working on group projects to work separately on a single document without having to deal with the complexities of a version control system. Also, users editing a collaborative script will be notified when a new user opens that script, thus allowing spontaneous real-time collaboration to occur. The owner of a collaborative script also has access to the version history of the script (a snapshot that is taken whenever the script is either run or saved), allowing them to roll back the contributions of a collaborator if they break the script.

4. USE CASE #2: LARGE ENSEMBLE LIVE CODING

4.1 Context

A second major collaboration feature we implemented was time-sync, which coordinated the playback of multiple EarSketch instances such that, for all scripts of the same tempo, the start of a measure will align to a beat on a common metronome. This feature was implemented with the goal of enabling large ensemble live-coding performance. Previous research with EarSketch had explored collaborative performance using the shared editor [18], but due to the lack of time synchronization between performers, only a single performer was tasked with playing the "master" audio. By allowing individual performers to play time-synchronized audio, we hoped to enable a co-located, synchronous collaborative environment where performers had a greater degree of independence and could collaborate more freely.

4.2 Design and Implementation

For synchronous real-time teamwork, we have implemented a quasi shared-clock system for multiple clients. By turning on the "Play Together" feature, and by using any ES script set to the same tempo (BPM), co-located clients can quickly synchronize their playback to time-quantized musical measures. This is achieved with the clock-synchronization algorithm found in the network time protocol (NTP) [13], that

$$serverClientOffset = ((ts - tc_1) - (ts - tc_2))/2$$

where tc_1 denotes the timestamp by the client upon querying, ts is the server timestamp at the query response, and tc_2 is the client's timestamp upon receiving the server response. By repeating this query (e.g., 30 times in our implementation) and taking the median, we estimate the stable time lag between the client and server with the assumed symmetric call-and-response communication lag. From this time offset, each client estimates the current server time, which is used as the basis for scheduling the playback at a timing quantized by the tempo and measure.

This "time sync" option deliberately does not automatically align the song locations and forms (though this is entirely possible by manually interacting with the DAW), in order to introduce several new performance / peer workflows. Particularly, co-located musical collaboration can now be split up into multiple ES scripts as opposed to the single shared script editing, and it is possible to combine the musical results in multiple ways, as discussed in the use-cases section below.

EarSketch provides a uniquely DAW-oriented programming environment. This brings opportunities in a loop-based as well as visualization-assisted live-coding performance [18].However, the DAW interface had also been a technical bottleneck for precise audio playback with the complexity of DAW operations and the limitations in the web-audio implementations in various browsers. Typically, EarSketch has to support the in-time (re)construction of a rather massive web-audio graph (easily up to hundreds of BufferSourceNodes and audio-effect nodes) while enabling various real-time user interactions, including:

- User operations on DAW, such as solo/mute, toggling effects, moving the play cursor, creating looped sections, and toggling a metronome
- Rendering recompiled scripts with new audio contents
- Toggling the time-sync option (new)

Previously, a solo live coder in EarSketch had to take these expensive operations into account as they may cause a gradual timing drift from the initial timing. In order to create a robust synchronization with the shared clock, it was essential to optimize the real-time rendering and playback engine. Here, we summarize the key strategies for better rendering and playback performance:

1) Tracking the AudioContext.currentTime for the past and future events: In EarSketch, a large number of audio clips across multiple tracks are scheduled to play (and stop) in sync according to the chosen starting position and loop / clock-sync configurations. While BufferSourceNode.start ensures precise playback timing, the scheduling process itself is a massive for-loop that needs to be completed sufficiently in advance of the actual playback. Simultaneously, we have to allow sudden changes and cancellations of the schedules by user actions. Taking a similar approach to Wilson's scheduling technique[1] we incorporate a window.setTimeout callback, which can be freely canceled as opposed to AudioNode.onended in some browsers, to calculate the next playback time using the offset between the AudioContext.currentTime and stored previous timestamps.

2) A queue-based rendering data management: EarSketch is a batch-processing system that (re)renders everything upon changes in the code, which is synonymous with how web-audio graphs behave. This creates a challenge in smoothly transitioning from the old rendered audio to the new one upon recompilation, especially with already scheduled play / stop times in each clip. Therefore, we need to schedule the new rendering data while keeping the old clips alive (until the newly-scheduled stop time). We manage these schedules in the queued rendering data based on the timing of musical measures, which proved to be more robust than immediately and continuously swapping the data.

The time-synchronized playback facilitates co-located exploratory collaborations where a live-coded composition task can be divided among multiple users. In addition to the existing pair programming mindset, it introduces a sound-oriented collaboration approach where each user explores the combination of different audio clips and edits different sections of a shared composition. Combined with the "Let Others Edit" feature above, we see a unique

[1] https://www.html5rocks.com/en/tutorials/audio/scheduling

opportunity of dynamic learning and creative activities in the classroom.

4.3 Applications

To gauge this potential, we piloted three impromptu "ensemble" patterns with an ensemble of 12 live coders. Each ensemble member had at least moderate experience with both EarSketch and musical performance.

4.3.1 Single script free-for-all

The first performance mode explored was of multiple users editing a single file. There were no set rules as to how the users should or should not edit the file, and users could render and play the audio at any time as they wished. All 12 machines played back the script simultaneously and in time sync. Users reported paying attention to their collaborators to make modifications in the music that would fit well, but also reported having trouble following the code and the music, because so much was changing with 12 simultaneous editors. An interesting consequence of this pattern was that, though there were only 12 users, there were sometimes more than 12 parts playing. This is because if a user had played music generated by a script and not re-run the script for a while, the script would have changed and would no longer reflect the music being played on other machines. Thus, users who had played the script more recently would be playing a different set of tracks.

4.3.2 Single script with limitations

In this pattern, as in the previous, all users were still editing a single script. However, this clip had a "template" song with 12 tracks (1 per user). The "rules" that were agreed upon were that a user would only edit his or her own track, and the user would "solo" their track, so that their computer only played audio of their own track. Users reported having an easier time following the code, but surprisingly, some said they had a harder time hearing the whole song - the large room in which the jam was conducted made it hard to hear the audio of all 12 computers at once to hear the "full" song. This was not an issue when each user was playing the "full" song on their own computer, effectively using it as a stage monitor.

4.3.3 Multi-file script for all

For the final pattern, we adopted a "multiple chat room" metaphor. Three collaborative scripts were shared among all 12 users, each containing their own separate music generating code. Users could move freely between scripts, editing and playing whichever script they chose. The goal was to create a hybrid of the previous two patterns - allowing the users to modify any part of the music, but also giving them a smaller, more manageable section of code in order to minimize churn. Users reported that dividing the collaborative environment into three scripts instead of one did not significantly reduce the complexity from the single-script free-for-all pattern.

5. DISCUSSION AND FUTURE WORK

Our informal study found that managing an improvisation session of 12 live-coders was difficult musically, but relatively manageable technically. For the most part, participants struggled to decide on musical actions, rather than struggling to figure out how to execute those actions with code. From a CS education standpoint, this is promising. Our end goal is to create a single platform whose context of use can fluidly switch between different modes of collaboration. We hope to make a tool that can be used to both teach programming concepts, and then allow expressive musical performance that reinforces those same concepts. Our future work is motivated by this inter-function fluidity.

5.1 Communication

We hope to improve strategies for communication and information sharing between remote collaborators. One specific feature we believe could improve communication is an in-EarSketch chat feature. Beyond simple verbal communication, a chat would allow collaborators to share snippets of code and make suggestions without cluttering the script being worked on with comments, and without having to swap back and forth between EarSketch and a secondary page. The chat feature could also be used in performance, allowing performers to share code without having to edit a common script.

5.2 Compilation

In our pilot 12-musician experiment, shared scripts were often challenging to use simply because they were so often in a syntax-incomplete state. In other words, one musician would be ready to run the code, but the code would not execute because another musician was editing another section of code. In past studies, we have avoided this problem by enforcing turn-taking [19], but this can be overly restrictive to collaboration. Live-coded computer music languages that support partial code execution, like SuperCollider, offer a more flexible model to consider.

5.3 Collaboration Tracking

While teachers have demanded increased collaborative support in EarSketch, they have simultaneously expressed concern about how those features will make it easier for students to plagiarize the work they submit in class. We plan to extend the existing version history feature in EarSketch to highlight which users made each change in a script, helping teachers to catch plagiarism, understand the different contributions of students towards group projects, and facilitate conversations about how students collaborated together and how they might work more effectively.

6. CONCLUSION

We hope to deepen student interaction with EarSketch through these new features and the activity types they enable. By providing a single tool that can be used both in educational and music performance contexts, both individually and collaboratively, we believe that the EarSketch environment can reinforce lessons learned in the class room and inspire curiosity outside of it.

7. ACKNOWLEDGEMENTS

EarSketch receives funding from the National Science Foundation (CNS #1138469, DRL #1417835, DUE #1504293, DRL #1612644, and IIP #1741045), the Scott Hudgens Family Foundation, the Arthur M. Blank Family Foundation, the Ruth L. Seigel Family Foundation, and the Google Inc. Fund of Tides Foundation. EarSketch is available online at earsketch.gatech.edu.

Any opinions, findings, and conclusions or recommendations expressed in these materials are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding bodies.

8. REFERENCES

[1] Astrachan, O. et al. 2013. CS Principles: Development and Evolution of a Course and a Community. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2013), 635–636.

[2] Barbosa, A. 2006. *Displaced Soundscapes: CSCW for Music Applications*. Universitat Pompeu Fabra.

[3] Collins, N. et al. 2003. Live Coding in Laptop Performance. *Organised Sound*. 8, 3 (Dec. 2003), 321–330. DOI:https://doi.org/10.1017/S135577180300030X.

[4] Kirkbride, R. 2016. FoxDot: Live coding with python and supercollider. (Hamilton, Ontario, Canada, Oct. 2016).

[5] Kirkbride, R. 2017. Troop: A Collaborative Tool for Live Coding. *Proceedings of the 14th Sound and Music Computing Conference* (2017), 104–9.

[6] Kluyver, T. et al. 2016. Jupyter Notebooks - A Publishing Format for Reproducible Computational Workflows. *ELPUB* (2016), 87–90.

[7] Lee, S.W. et al. 2011. Collaborative musical improvisation in a laptop ensemble with LOLC. (Atlanta, Georgia, USA, Nov. 2011), 361–362.

[8] Lerman, L. and Borstel, J. 2003. Critical Response Process. *Takoma Park, MD: Dance Exchange*. (2003).

[9] Mahadevan, A. et al. 2015. EarSketch: Teaching computational music remixing in an online Web Audio based learning environment. *Web Audio Conference* (2015).

[10] McCartney, J. 2002. Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal*. 26, 4 (Dec. 2002), 61–68. DOI:https://doi.org/10.1162/014892602320991383.

[11] McKiln, T. et al. 2018. Authenticity and Personal Creativity: How EarSketch Affects Student Persistence. (Feb. 2018), 987–992.

[12] McLean, A. and Wiggins, G. 2010. Tidal – Pattern Language for the Live Coding of Music. *Proceedings of the 7th sound and music computing conference* (2010).

[13] Mills, D.L. 1991. Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications*. 39, 10 (Oct. 1991), 1482–1493. DOI:https://doi.org/10.1109/26.103043.

[14] Ogborn, D. et al. 2017. Estuary: Browser-based Collaborative Projectional Live Coding of Musical Patterns. *International Conference on Live Coding (ICLC) 2017* (2017).

[15] Roberts, C. and Kuchera-Morin, J. 2012. Gibber: Live Coding Audio in the Browser. *ICMC* (2012).

[16] Rohrhuber, J. and de Campo, A. 2009. Improvising Formalisation — Conversational Programming and Live Coding. (2009).

[17] Sun, C. and Ellis, C. 1998. Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements. *Proceedings of the 1998 ACM conference on Computer supported cooperative work* (1998), 59–68.

[18] Xambó, A. et al. 2016. Challenges and New Directions for Collaborative Live Coding in the Classroom. *International Conference of Live Interfaces (ICLI 2016). Brighton, UK* (2016).

[19] Xambó, A. et al. 2018. Turn-taking and Online Chatting in Remote and Co-located Collaborative Music Live Coding. *Journal of the Audio Engineering Society*. 66, 4 (2018).