SHARP: Exploring Version Control Systems in Live Coding Music

Daniel Manesh
danielmanesh@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Douglas Bowman Jr.
drewb00@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Sang Won Lee
sangwonlee@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

ABSTRACT

Version control systems, which have proven essential for software engineering, can also provide value to creative and artistic practices. In this paper, we explore version control in the creative domain of live coding music, a generative performance practice where programmers edit and run code live to generate audiovisual artifacts. To that end, we developed SHARP, a lightweight version control system that live coders can use during performances as well as in preparation or practice sessions. We conducted a user study where live coders used SHARP for several weeks, wrote diary entries reflecting on their sessions, recorded a performance using SHARP, and participated in exit interviews. We found that SHARP enabled participants to engage with musical form on the fly in novel ways. In addition, the study revealed multifaceted perspectives on how and when versioning can be useful in the context of live coding. Our results inform the design of versioning systems for live coding and more generally for performance and generative arts practices.

CCS CONCEPTS

• **Applied computing** → **Sound and music computing**; **Performing arts**; • **Software and its engineering** → **Software configuration management and version control systems**; • **Human-centered computing** → User studies.

KEYWORDS

live coding, versioning, exploratory programming, creativity support tools

ACM Reference Format:
Daniel Manesh, Douglas Bowman Jr., and Sang Won Lee. 2024. SHARP: Exploring Version Control Systems in Live Coding Music. In *Creativity and Cognition (C&C '24), June 23–26, 2024, Chicago, IL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3635636.3656195

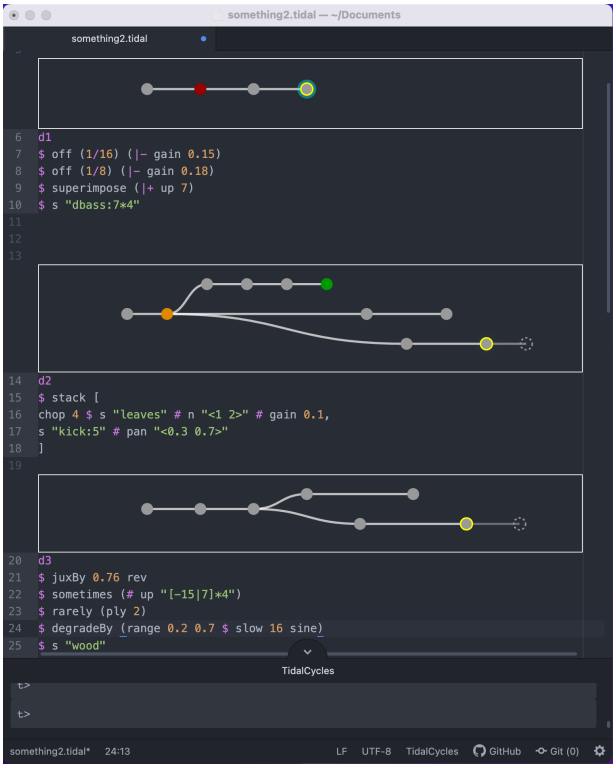

Figure 1: SHARP is a lightweight version control system designed for live coding music. SHARP embeds version trees in the text editor above the corresponding code blocks. The version trees help live coders understand the evolution of each pattern while also allowing quick retrieval of previous versions by clicking on nodes.

1 INTRODUCTION

Engaging with past artifacts is essential for various creative practices, from tracking changes in writing [32], managing multiple versions of files [16], taking photos of physical prototypes [62], creating a visual history of website designs [31], or reusing existing designs for new products [10]. Researchers have studied how artists and creators document their creative process, studying how practitioners use version control systems (VCS) — either digital or not — and what the motivations are behind using (or not using) VCS [10, 47, 62, 75].

VCS have been extensively studied in programming and have long been used in practice [38, 54, 76, 77]. In a software development context, typical goals for VCS include collaboration, risk isolation (e.g., reverting changes), and safeguarding the production code from development [51]. However, the needs and functions of VCS in a creative and artistic context can deviate from the needs of traditional software engineering [62], and researchers have begun examining version control in a creative context [55, 58, 62].

In this paper, we explore version control in the domain of *live coding music*, a highly dynamic performance practice in which a programmer writes code to generate audiovisual artifacts, typically generative music, in live performance [15, 40]. Although live coding music is mediated through code, a common and extensively studied

C&C '24, June 23–26, 2024, Chicago, IL, USA

ACM ISBN 979-8-4007-0485-7/24/06
https://doi.org/10.1145/3635636.3656195

medium for VCS, the unique characteristics of the performing arts can influence how a VCS should be designed to support exploratory programming [2] and tracking versions of artistic artifacts [10]. In particular, live coding is performed *live*, in front of an audience, and has elements of both composition as well as improvisation [49].

To understand how VCS can be designed to support a performing art practice, we developed SHARP (Figure 1), which stands for **S**tate-**H**istory **A**ugmentation for **R**apid **P**rogramming. SHARP is an editor plugin designed to work with a popular live coding language, Tidal Cycles [41]. SHARP aims to support rapid and near-effortless interaction so that live coders can document and retrieve previous versions from their execution history under the time pressure of a live performance. To that end, SHARP introduces lightweight, interactive *version trees* that update automatically as the user edits and runs their code. Users can quickly navigate versions by simply clicking on nodes in the version trees. SHARP creates a separate version tree for each *pattern* in Tidal Cycles, a musical unit that can be thought of as analagous to a single instrument in an ensemble. This way, live coders can track the evolution of each individual pattern separately, and can mix and match different versions of musical patterns for enhanced musical expressivity.

Using SHARP as a probe [26], we conducted a user study with live coding musicians ($n = 6$) to answer the following research questions.

- How does a VCS embedded in a musician's interface affect their performance and preparation for live coding?
- How should a VCS represent and preserve its version history for live coding?

The participants used SHARP over several weeks, wrote diary entries reflecting on their sessions, recorded a short performance using SHARP, and participated in exit interviews. Our study found insights into how a version control system designed for live coding could expand musical expressivity and how the system created novel ways for musical sensemaking (e.g., a version history as a musical score for communication and analysis). In addition, we found multifaceted perspectives — sensemaking, reconstruction, code organization, and temporal/sequential ordering — regarding the utility of versioning in the context of live coding.

2 RELATED WORKS

2.1 Tracking Fine-Grained Changes in Code

While traditional VCS like Git and Subversion have proven valuable, they are relatively slow to use, and work best at tracking large-scale changes in a collaborative setting [44]. Researchers have found that it can also be valuable to track smaller code changes, like insertions and deletions in a text editor [23, 44, 72]. As live coding is characterized by frequent updates to code, we are more interested in these *fine-grained* code changes.

Keeping a fine-grained code history is typically done by keeping track of *actions* (i.e., individual edits) and/or *states* (i.e., the program state between edits) [24]. Modern text editors store a linear timeline of edit *actions*, allowing users to undo/redo changes [73]. While useful, a linear edit history is insufficient for some backtracking tasks [73], and researchers have explored more flexible methods which allow users to selectively undo edits localized to a specific region of code [44, 71].

Keeping track of program *states* can be useful for exploring and comparing alternatives [45]. This is particularly appropriate for *exploratory programming* practices, such as data science and creative coding, which are characterized by rapid experimentation and evolving goals [3]. Without specialized tools, exploratory programmers often resort to ad-hoc versioning techniques like commenting out code, copying code snippets, and duplicating files [27]. Systems which track program states allow programmers to compare variants which may be scoped at the file level [22], the code-block level [27], or even at the level of both code blocks and individual lines hierarchically [29]. Some of these systems rely on the user to indicate when they want to keep track of a version [22, 27, 67]. Others track versions proactively, automatically creating a version when the user runs code [28, 29]. Proactive versioning systems may make use of features like bookmarking [29] or search [28] to ensure users can find the versions they care about.

Systems tracking program history must consider how to best present that information to the user. Common approaches include timelines [72], tree visualizations [27], and even just lists of edits/states in reverse chronological order [29]. More complex visualizations can help grok the evolution of the code over time [70] or capture nuances with backtracking and other user interactions [45].

2.2 Version Control and Creative Work

Rich history-keeping is considered an important design principle for developing creativity support tools [61]. The way creative practitioners engage with past versions of their work is nuanced and differs from how software engineers might engage with previous versions of their code. For example, artists may treat past versions of their work as palettes; thus version histories may not represent linear progress, and previous versions may be just as valuable as the latest [62]. Complicating matters, Nicholas et al. found that creatives may employ "strategic forgetting" by choosing to capture versions of their work with lower-fidelity, or foregoing recording version histories at all [47]. Similarly, Li et al. found that some digital artists choose to purposefully forego using the *undo* feature so as not to break their flow [37]. Creatives are not a monolith, and we expect live coders to have a variety of attitudes towards versioning in their creative practice.

Recent work has looked specifically at VCS for creative coding. One example is Quickpose, a VCS designed for creating visual artifacts with Processing [55]. With Quickpose, users manually specify when to record versions or create branches, and they can interact with a graphical representation of their version histories by moving nodes and adding text labels. Another recent work is Spellburst, a VCS for creative coding which similarly allows users to interact with a graphical representation of their program versions to explore new variations [1]. In contrast to these two systems, SHARP is designed to be lightweight and simple enough to be used during a live performance.

2.3 Live Coding Music

2.3.1 ***The Practice of Live Coding.*** Live coding is a creative practice where a performer continuously modifies and runs code to create a live audiovisual performance [15, 40, 66]. From its inception, live coding has been perceived as improvisational [15],

though improvisation during performances may be limited in some cases [42]. In practice, there is a spectrum of how much a live coder has prepared in advance: some completely improvise starting from a "blank slate", some come with pre-composed snippets of code to experiment with, and some plan out their entire performances with pre-written code [4, 17].

During live coding performances, a common practice is for live coders to share their screens with the audience [4, 7, 13]. In addition to providing a particular aesthetic [8], this practice can give the audience insight into the underlying musical process of algorithmic composition [8, 56] and can even be used as a communication channel to the audience (e.g., through code comments) [34].

2.3.2 ***Tools to Support Live Coders.*** Many tools have been developed to assist live coders. Some tools may make it easier to work with certain types of data, like drag-and-drop mechanisms for importing audio files [57] or graphical widgets to facilitate working with streams of sensor data [19]. Other tools help visualize the state of the program, for example, displaying real-time values for variables [35, 57] or showing the current progress of audio loops [63]. These tools and visualizations not only can help the performer [63], but can provide visual interest or even explanatory power to the audience [43, 52]. This is of particular interest, as it can be difficult for the audience to understand what's going on, even when the code is projected [6, 56].

Few existing tools support versioning for live coding. One tool, CodeBank, allows performers to run snippets of code locally to preview the resulting sound before "publishing" the new code version for the audience to hear [30]. CodeBank focuses on safety and collaboration, and does not provide an interface for easily navigating between versions. The live coding language SuperCollider has a History class which records code executions and can be used to programmatically navigate the execution history.[1] While the History class can be used for post-hoc study [49], it does not provide a graphical interface and may be too cumbersome for a live performance setting.

3 DESIGNING A VCS FOR LIVE CODING

3.1 Tidal Cycles — A Programming Language to Dance to

Among the many live coding languages available, we chose to support Tidal Cycles (Tidal for short), a live-coding language written in Haskell [41]. We chose Tidal because it is one of the most widely used live coding languages and has an active online community. Tidal has integrations with several code editors, including Pulsar, the code editor which SHARP is built for. Figure 1 shows Pulsar with the SHARP package running.

Tidal performances are typically composed of multiple code blocks, as live coding typically involves multiple tracks/instruments layered on top of one another. Typically, each block has code for a pattern that plays a loop (or a cycle) of sound (see the code in Figure 2). A *pattern* in Tidal Cycles is similar to a track in a digital audio workstation (DAW) or an instrument in a band/ensemble. A live coder can *evaluate*, i.e., execute/run, their code by highlighting a region of code they want to evaluate and pressing Ctrl/Cmd+Enter. If no line is highlighted, it will run the current code block that the text cursor is in. Although flexible, the style of music for Tidal is typically generative, rhythmic, and electronic.[2]

[1] See the documentation at: https://doc.sccode.org/Classes/History.html

3.2 Design Principles

We consider the following four design principles for SHARP and explain how they are informed by our target domain of live coding music.

(DP1) Keep Interactions Simple Interactions with a VCS for live coding need to be as simple as possible. The act of live coding is closer to live composition than to playing an instrument [40] and can require a high cognitive load [21]. The VCS should not disrupt the creative flow, which is vital for capturing and exploring musical ideas [58].

(DP2) Support Exploration of New Ideas Live coding is an improvisational music practice where serendipity and novelty are valued [4, 14, 15]. Thus, a VCS should support exploration, for example, by easily recombining different ideas from past program states.

(DP3) Facilitate Big Changes While gesture-based musical instruments can create sounds almost immediately, live coding requires time for crafting algorithms and precisely typing out code [49, 65]. Thus, it is typically difficult for live coders to pull off large changes [6], and live-coded music commonly changes gradually over time. A VCS for live coding presents an opportunity to more easily enable large changes through quick version navigation.

(DP4) Show the Program State When live coding, it may take the performer time to write a new pattern, and the code on the screen may no longer match the sound that is playing. Swift et al. refer to this as a discrepancy between the code that is associated with the currently generated sound, *the State of World*, and the code that is currently on the screen, *the State of Code* [63]. Because a VCS for live coding deals with the code evaluation history, it is a good opportunity to help clarify the relationship between the code on the screen and the sounds that we hear.

4 SHARP: SYSTEM DESCRIPTION

SHARP is a VCS designed for the live coding language Tidal Cycles and built as a plugin for the Pulsar text editor. In the following section, we describe the features of SHARP and how they related to the design principles in 3.2.

4.1 Automatic Versioning from Code Execution

In SHARP, the version history of a pattern is visualized in a *version tree* structure. Version trees are situated inside the text editor, just above the corresponding block of code (see Figure 2). Nodes, as represented by a circle, are *automatically* created whenever a live coder *evaluates* a particular pattern. This automatic version control differs from common version control systems where a programmer must choose when to commit their code and create a version. By

[2] For more context on live coding and Tidal Cycles, refer to the following TED Talk by Alex McLean, the creator of Tidal Cycles: https://youtu.be/nAGjTYa95HM?t=461. For a demo performance using SHARP, refer to the following recording by the first author: https://youtu.be/EuumU702Ppw.

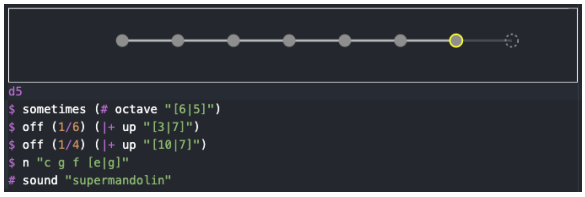

Figure 2: SHARP maintains a version tree for each Tidal pattern, which is placed above the corresponding block of code. Here, there are a total of seven nodes created for the pattern "`d5`". The rightmost node is a "ghost node", which is yet to be committed to the version tree, and which corresponds to the code seen here.

removing this decision, we prevent an additional cognitive burden for live coders during live performance (DP1).

4.2 Per-pattern Version Tree

In SHARP, a version tree is created for each pattern, a Tidal abstraction somewhat analogous to a single instrument (see 3.1). While many version control systems track the history of changes for an entire project (e.g., a repository for GitHub), we instead opt for per-pattern version trees for two reasons. First, keeping per-pattern version trees creates a more compact version tree than keeping a single, global tree, and thus is less overwhelming (DP1). Second, per-pattern version trees allow live coders to explore versions for each pattern independently, allowing them to mix and match different versions of patterns with flexibility (DP2). For example, if a live coder wants to mix a drum pattern submitted in the past without changing a drone sound being played in another pattern, they can click the node of a drum pattern that they want to retrieve and evaluate the code without affecting the drone sound.

4.3 Navigating the Version Tree

The tree graph is not just a visualization of a pattern history but also an interface that a live coder can use to *retrieve* or *preview* a previous version of code.

4.3.1 ***Click to Retrieve Code.*** Clicking a node will replace the current code block with the previous version of the code that was stored in that version. Notably, clicking a node does not automatically run the code block. Although this means one more interaction is needed if the user wants to reevaluate the pattern, retrieving the code text only without running it can be useful in multiple ways. First, a live coder's intention may not be to replace the current pattern with the previous version but to create a branch from the previous sound. For example, instead of returning to a bass riff pattern that was previously played, one may want to make a new variation of the original bass riff pattern (e.g., with a distort effect). Second, a live coder can click multiple nodes across different version trees and wait to evaluate them all together. In this way, they can change multiple patterns simultaneously instead of sequentially, facilitating larger, dramatic changes (DP3).

4.3.2 ***Hover to Preview and Diff.*** Hovering the mouse over a node will preview how the block of code below will change if the hovered-over variation is selected. If the user moves the mouse outside of the node, the code will revert back to its old state. We found that hovering in and out of a node was a simple and effective way to diff the two versions of code, as pattern code is typically short enough to fit on one screen without needing to scroll (DP1).

Figure 3: A live coder can create a branch from any node by clicking a node and revising its code. Here, the second node from the left has two branches (e.g., two musical variations). The horizontal placement of the nodes corresponds to their creation time, with newer nodes appearing to the right. The selected node (highlighted yellow) is the node that was added most recently.

4.4 Visual Cues for Rapid Version Control

SHARP is designed to be usable even under the time pressure of a live performance. Thus, SHARP makes use of visual cues to help live coders control and comprehend code more effectively at a glance.

4.4.1 ***Chronological Ordering.*** In SHARP, whenever a new node is created, it is added to the right of all the existing nodes in the version tree, as seen in Figure 3. In this way, the version trees act as a timeline for when each version was created. Live coders can use their memory of when they created different versions to help intuitively locate them in the tree (DP1).

4.4.2 ***What do we See? What do we Hear?*** SHARP has separate decorations for nodes based on the state of the program. A pulsing blue border indicates the most recently evaluated version of a pattern, i.e., what we currently *hear* (*state of the world* in DP4). Note that the node we hear might be different from the currently selected node; selecting a node will change the code (*state of the code* in DP4) in the code block but does not affect the music because it is not automatically evaluated. Selecting a node in the tree will highlight the node with a bright yellow border (see Figure 4). Finally, once the user starts revising code, a *ghost node* with a dotted-line border will be created (the rightmost node in Figure 2). The ghost node anticipates where the node will be added and indicates that it is yet to be submitted. Once the code is evaluated, the ghost node transitions to a solid node, indicating that it is now a saved version that can later be retrieved (DP4).

4.4.3 ***Tagging Nodes for Easy Retrieval.*** A live coder can tag nodes with a color to distinguish them from others. For example, for a particular bass riff they know they want to repeat later in the piece, a live coder can click the node with the option key pressed to tag it with a color (see Figure 4); clicking multiple times cycles through several color options. Once the tree grows, it will be helpful for the musician to recognize previous versions at a glance. While it is common in VCS to annotate versions with text (e.g., commit messages in GitHub), we do not ask (or allow) live coders to provide a text label, as this may be too time-consuming or distracting for a live setting (DP1).

Figure 4: SHARP uses several visualizations to help live coders comprehend the current state. 1) The node highlighted in yellow is the currently selected code for this pattern; the corresponding code will be available in the code block. 2) the rightmost node with a pulsing blue color is the version that was most recently executed, i.e., what is currently playing. 3) The purple, red, and orange nodes have been manually tagged by the user so they can easily recognize them in case they would like to revisit them.

5 STUDY AND METHODS

Our study is designed to understand how SHARP influences the practice of live coding musicians and more generally how live coders may find VCS useful in the context of live coding. To that end, we conducted a field study in which we asked live coding musicians to use SHARP and reflect upon their experience. The study was approved by the Institutional Review Board of the authors' university.

5.1 Participants

We aimed to recruit live coders who were comfortable with Tidal, the language SHARP was designed for. We required that participants had multiple performance experiences, as we wanted participants to have an informed opinion on using SHARP in a performance setting. We also considered performance experience a reasonable proxy for general live coding experience.

We recruited our participants through advertisements posted in two live coding communities: the Tidal Cycles Discord Server and the TOPLAP Live Coding Discord Server. We also recruited one participant through the snowball sampling method. Participants filled out a short screening survey about their experience live coding, and we selected participants who matched our criteria. In total, we recruited six live coding musicians, and their participation was compensated with a $75 electronic gift card or the equivalent in their local currency. Participants' background information is available in Table 1.

5.2 Study Procedure

We designed our study to emphasize ecological validity, giving participants a chance to use SHARP outside of a lab setting. To that end, we asked participants to go through the following steps.

5.2.1 ***Step 1: Onboarding Session.*** For each participant, we conducted an hour-long onboarding session via Zoom. This included an interview, a brief tutorial of SHARP, and the study instructions. The initial interview included questions about their typical live coding practices and did not focus on VCS; we therefore excluded the initial interview from our analysis and plan to report those results elsewhere. The first author shared his screen during the tutorial, explained what SHARP did, and then walked through the various features. While 12 participants completed the onboarding process, only 6 of them continued on to complete the study.

5.2.2 ***Step 2: A Self-paced Activity.*** We asked participants to use SHARP for 3 to 5 sessions, which participants completed on their own over the course of 1 to 4 weeks. At the end of each session, we asked participants to fill out a diary through an online survey that we created. The survey had one open-ended text response with some guiding questions, available in Appendix A.1. The purpose of the diary was to prompt reflection and also to serve as a memory aid we could reference in the exit interview.

In addition to the diary entries, we asked the participants to create a screen recording of a short performance. The purpose of the recording was both to give them a realistic goal to work towards during their sessions and also to provide a performance-like setting where they could use SHARP.

5.2.3 ***Step 3: Exit Interview.*** After confirming they completed step 2, we conducted semi-structured exit interviews with each participant, which typically lasted one hour. During the interview, we had participants reflect on their usage of SHARP in their sessions and performance. We also had them suggest improvements to SHARP and speculate on when and how VCS could be valuable for live coding. The list of guiding questions is available in Appendix A.2.

5.3 Data Collection and Analysis

The first and second authors conducted the onboarding sessions (step 1) and the first author conducted the exit interviews (step 3). We had a total of 26 diary entries, though we only analyzed the 23 entries from the 6 participants who completed the study. We analyzed transcriptions of the exit interviews, the diary entries, and the performance recordings to identify emerging themes using thematic analysis [12]. Specifically, the first author conducted open coding of the transcriptions and diary entries, and the first and last author together conducted axial coding, grouping and merging the codes and constructing themes. The authors discussed the results until they agreed on the set of themes, which correspond to the subsections in 6. Lastly, we inspected the performance recordings as well to supplement the themes as needed.

6 RESULTS

Overall, participants had a positive experience using SHARP for the study. P1 and P5 appreciated that SHARP made them think about their practice in a different way, while P3 liked that SHARP sped up and improved their normal process. Some participants expressed interest in using SHARP in the future, including P3 who enthusiastically expressed "*I really love SHARP and I think I will never use Tidal Cycles again without it*".

6.1 When is Versioning Useful?

Participants had differing opinions regarding the contexts where they found SHARP to be useful. P3 and P6 appreciated SHARP in both a performance context and in composing or exploring ideas in a non-live setting. Other participants preferred one setting over the other. For example, P2 particularly appreciated SHARP for a performance setting, but thought it was not helpful during initial

Table 1: The background information of study participants.

Participant	Age	Gender	Number of Performances	Number of Years Live Coding	Self-assessed expertise	Country
P1	25	Man	3-9 performances	2.5 years	Proficient	USA
P2	43	Man	10+ performances	3 years	Proficient	Australia
P3	41	Man	3-9 performances	3 years	Competent	Italy
P4	42	Man	3-9 performances	5 years	Proficient	USA
P5	30	Woman	10+ performances	5 years	Proficient	France
P6	28	Man	10+ performances	5 years	Advanced Beginner	USA

exploratory phases in their practice sessions. On the other hand, P1 felt the opposite: they appreciated SHARP for initial explorations of an idea, but felt they would not use SHARP for a performance. P1's primary reason was that they did not want to backtrack during a performance. This sentiment was shared with P5, who said of their improvisational performances "*If it's already been shown to the audience, I'm not going back*". P4 had different reservations about a performance setting: "*[SHARP] would be fun to use in a live setting if I wasn't such, like, a slow coder*". Notably, P4 reported using primarily pre-written code in their performances. They expressed interest in using a versioning system like SHARP in a live setting with pre-written code already loaded in version trees.

6.2 Navigating the Version History

> *(P6) I was trying to think of it as an improvisational or compositional resource, like same as arpeggiating a chord or duplicating a voice or something. Now I am able to go back in time by clicking on a node.*

Participants used the version navigation in SHARP in four ways: to backtrack to a safe point, to explore new combinations of versions, to incorporate musical form into a performance, and to repeatedly compare variants. Participants also imagined a new interaction of being able to automatically navigate through the version history during a performance.

6.2.1 ***Backtrack to a Safe Point.*** Both P2 and P3 liked having "safe points" readily available in SHARP's version trees. P2 appreciated not worrying about which versions of their code they should save, as SHARP saved them automatically. P3 felt that SHARP made them more comfortable experimenting with complex patterns.

> *(P3) I like that you have, like, a safe point, let's say. And then the next pattern can be, like, very fucked up in terms of complex rhythm, and then you can go back to the thing that was working. [...] It's good, because sometimes you find yourself in a situation where everything is really crazy, and it's good for a short time, but then it's too much.*

6.2.2 ***Realizing Musical Form.*** Some participants found that SHARP's version history allowed them to more easily incorporate musical form into their performances.

> *(P2) Song form ideas are the weakest part of my performing for sure. [...] It would be nice to be able to go back in this fashion in a regular way. [...] But to this point, I haven't had a great mechanism for doing it.*

SHARP gave P2 a way to easily backtrack in a performance without worrying about which states to save ahead of time. They made use of this in their musical performance: they did an improvisation with a particular form in mind, and were able to easily achieve it using SHARP's state navigation.

P1, P2, and P6 all found SHARP useful for the common approach of building up complex patterns and then winding them down at the end of a piece. P6 demonstrated this technique in their performance video, where they ended their piece by gradually backtracking multiple patterns to earlier, less complex versions. P6 described previously using this approach in performances by commenting out code, but felt that commenting out code was more error-prone than using SHARP.

> *(P6) sometimes I mess it up because I forget, out of all the lines in my* `do-block`*, which are the ones that I should be commenting out and which are the ones that I should leave.*

6.2.3 ***Compare and Curate for Composition.*** SHARP's version trees gave participants a way to easily compare multiple versions and choose which ones they wanted to use.

> *(P1) It helped give me a bunch of easily accessible versions of patterns in a concise format which is nice, and from there I could take the versions I liked and make something more structured.*

> *(P5) It's more [useful] like when you you've come up with an idea and you want to find the best version of it. So you have to compare between menu changes rather than just going by a stream of consciousness.*

SHARP allowed participants to easily toggle back and forth between different versions to compare their audio output. While this use case is unlikely to be appropriate for a live performance, it can be useful in preparing for one. Additionally, this can be useful for developing a fixed media track to be recorded rather than performed, a use case which both P3 and P5 considered well-suited to SHARP.

6.2.4 ***Supporting Exploration for High-level Improvisation.*** Participants considered navigating the version trees in an exploratory manner to try out new combinations and sequences of program versions.

> *(P1) I had serendipity because I was sometimes just randomly clicking on nodes and seeing what popped up.*
>
> *(P6) if you are just clicking through the nodes and hitting enter, like, the order in which you hit the nodes could be also improvised. So it would be more, like, abstract layers of improvisation.*

This "*abstract layer of improvisation*" (P6) was also hinted at by P5, who liked the idea of being able to easily *"mix and match"* (P5) versions to get new combinations.

6.2.5 ***Automatic Version Navigation.*** Although not currently supported by SHARP, three participants (P2, P3, P6) brought up a new idea of being able to automatically play a sequence of versions from SHARP's version trees.

> *(P2) You could potentially be able to go through and select a series of nodes and go "replay this". [...] actually using it as a playback mechanism during the live performance.*

With such a feature, live coders could schedule code changes for the future, essentially allowing live collaboration with their past selves in a performance setting. P4 and P6 imagined similar automatic mechanisms, but also considered being able to add randomness or even AI to affect the order that the versions are replayed.

6.3 An Additional Layer for Sensemaking

Keeping an organized version history with SHARP provided an extra representation, other than code itself, for live coders to reason about their creations.

6.3.1 ***Sensemaking in Live Performance.*** Seeing SHARP's version trees during a performance can be beneficial to the performer. For example, P1, P4, and P6 all suggested that SHARP might help prevent them from losing their place during a performance. SHARP's version trees helped participants understand what they had already done during a performance, and also served as reminder of what *could* be done in the future.

> *(P1) It was easy to keep [branching] in mind because I could see, I could visualize the tree [...] I had a general idea of, like, where it had been.*

In addition to helping the performer, the participants thought version trees would help the audience. With SHARP, the audience can look at the highlighted node in the tree to determine whether they were hearing a previously-played version of a pattern. P4 and P6 both appreciated this use case, and thought that SHARP's version trees provided a good way to see how a piece evolves over the course of a performance. We could also confirm this effect by inspecting P6's performance recording; the researchers could clearly see them "winding down" their piece towards the end by navigating to earlier nodes in their version trees.

Finally, P5 saw promise in adapting SHARP's version trees for use in a live collaborative setting. They imagined augmenting the version nodes with provenance information to help understand who made which changes.

> *(P5) From my experience jamming on Estuary [3] [...] sometimes you're like, who? Who did this?*

6.3.2 ***Reconstructing Musical Ideas.*** The participants found that version histories can provide a means to reconstruct a piece with varying levels of fidelity (i.e., exactly or partially). Even though SHARP did not persist version trees between sessions, participants imagined how persistent version trees might be useful for musical composition.

Resuming Context The structure of the version trees can act as a visual cue to help live coders restore their work context. For example, P3 reported the tree structure aided them when coming back to work on a composition after a short coffee break away from the computer. On a longer timescale, P1 mentioned sometimes returning to old ideas after a year or more, and thought that the extra structure from version trees would help them remember what they had in mind.

Reviewing Performances Participants considered that SHARP's version trees could be useful to document and review one's own performance. This could be useful for identifying and learning from common mistakes (P2) or for reviewing a composed (i.e., non-improvised) piece to look for opportunities to improve it in the future (P6).

Sharing Performances In addition to reviewing one's own performance, version trees could be useful for sharing a piece with others. P6 likened the version trees as a type of score, and thought something like SHARP would have been helpful during their previous collaborations.

> *(P6) SHARP would be very useful because it would be useful as a communication tool. [...] It would be useful to show your ideas to other people and explaining to them what you want to do.*

P6 and P3 both imagined features to edit or remix pieces shared via version histories.

Version History vs. Screen Recording Notably, one could also replay a performance simply by recording one's screen. However, a screen recording does not allow for copying code or stepping through the program versions at one's own pace. Furthermore, as P6 pointed out, screen recording can be resource-intensive, and combined with the other computational demands of live coding, it is not always a viable option. On the other hand, screen recording captures the exact code execution order and timing, information which is not currently available in SHARP (see 6.5).

6.4 Moving Messiness from the Code to the Trees

6.4.1 ***Versioning Can Simplify Code.*** Because SHARP automatically records the version history, participants could forego ad-hoc versioning techniques like code duplication or commenting out code. As P1 said: "*I repeat a lot of code without SHARP*".

P1, P3, and P4 commented on how cleaner code with SHARP could be beneficial in a performance setting, as having less code would reduce the mental load on both the performer and audience.

[3] Estuary is a collaborative live coding platform [50]

> *(P4) my code files tend to be, like I said, pretty long. And so [SHARP] might be an easy way to sort of, not have to scroll as much and potentially lose your place.*
>
> *(P3) Aesthetically, maybe it's better [for the audience] to see [SHARP's interface] than to see like 50 lines of commented-out code.*

Notably, two participants brought up drawbacks to having cleaner code. P1 and P2 both felt that there was some benefit from duplicating several versions of their code and being able to scroll past and glance at the versions in a textual format.

> *(P1) Maybe I'm trying to go back to one [version], but I see another that seems interesting and I'll try that.*

While P1 was concerned with serendipity, P2's main concern was how to most quickly find a specific previous version. They felt this was easier to do by scanning the actual text and felt *"the nodes are very opaque" (P2)*.

6.4.2 ***Version Trees Can be Messy.*** Even though SHARP often simplified the code in the text editor, it introduced a new type of messiness in the version trees. All participants except P3 mentioned that they at some point had too many nodes to easily use the version trees, or that they expected to run into that issue in longer sessions. Participants' suggestions for dealing with unruly version trees fell under two strategies: 1) annotation and search mechanisms to find useful nodes; and 2) strategies to edit the tree to a more manageable size. We elaborate in the following sections.

6.4.3 ***Foraging in the Version History.*** SHARP's node tagging feature was designed to be a lightweight way to mark versions that could be used in a performance setting. All participants made use of the tagging feature to color specific nodes. As intended, participants mentioned tagging nodes they felt were good and that they might want to return to.

> *(P2) I would think about going, building up to a section. And thinking of that section as complete, I'd mark it green, then I'd move on.*

P4 and P6 wanted the ability to further annotate nodes with short text labels, a feature that P1 considered possibly useful and that P5 thought was unnecessary unless the version history was extremely large. P6 envisioned the text labels could be useful for a search feature, which they supposed would not make sense for a live performance, but could be useful if composing or practicing.

P2 considered automatically augmenting the nodes with additional information to help make sense of each node at a glance. One signal they suggested was the duration each version was active, which could be useful, for example, to quickly locate the longest-running versions.

6.4.4 ***Managing the Graph Structure.*** *All* of the participants thought it would be useful to edit the graph by removing nodes in some way. P6 primarily imagined cleaning up the graph as a useful activity if they could share their version trees. The other participants imagined that pruning the graph as they went along would be useful for keeping the graph manageable during their own sessions. Often, it was easy for them to identify nodes they would like to delete.

> *(P4) Getting from one idea to another idea, there was like this really, really long string of nodes. And I would have loved to have just been, like, able to just delete some of them.*

Rather than deleting nodes, related nodes could also be merged together. P5 described a situation where versions that differed in one numerical parameter could be automatically merged.

> *(P5) There were a lot of recorded states that were mostly redundant because I tend to reevaluate the code very often and like sometimes it's just a change of one number.*

P5 thought automatically merging the nodes was feasible, but preferred a manual merge operation, expressing wariness of *"the choices of the algorithm"* (P5).

Finally, rather than deleting nodes, a system could allow the user to proactively decide whether or not to record a version when running code. This feature was specifically requested by P4. P1 found a bug in SHARP where they were able to run code with or without versioning by using different hot keys, which they found convenient for repeatedly iterating on a pattern. P2 and P5 preferred to *"dismiss intermediate versions" (P5)* after the fact, so as not to interrupt their flow while exploring.

6.5 How Should We Version?

SHARP's versioning model is not powerful enough to exactly reconstruct a performance based on version trees. This was by design: SHARP was meant to prioritize intuitiveness and quick usage over more comprehensive record-keeping functionality. As we had hoped, our participants were able to easily understand and use SHARP as a versioning system. Still, tensions arose from the versioning capabilities missing from SHARP, which can be categorized by four missing types of information:

Pattern Cotemporality When we track a version for one pattern, we do not keep track of what versions were active for other patterns. So, if one pattern has versions A and B and another has versions C and D, we do not know if pattern A ever played at the same time as patterns C or D.

Execution Sequence and Timing Our versioning model does not record instances of backtracking, so it is impossible to reconstruct the exact execution order from the version trees. For example, take three versions of a pattern: A, B, and C. If the versions are executed in the order $A \rightarrow B \rightarrow C \rightarrow A$, the version tree in SHARP will be indistinguishable from the execution ordering $A \rightarrow B \rightarrow C \rightarrow B$. Furthermore, we do not record timing information for when a pattern is executed.

Non-pattern State Code that does not define a pattern in Tidal is not versioned. Thus, tempo changes or variable assignments are not tracked in versions. P3, P5, and P6 wanted this feature.

Sub-pattern Versioning The smallest level of scope we have for versions is a Tidal pattern. We do not, for example, track individual lines of code within a pattern, though P5 expressed interest in such a feature.

6.5.1 ***Pattern Cotemporality.*** SHARP does not keep track of which set of pattern versions were active at the same time. P2 and P3

suggested a *global tree* which would encode exactly this information in each node. They envisioned this as a more convenient mechanism to backtrack across multiple patterns at the same time.

> *(P2) There might be times when you want to take the composition back to a specific spot, instead of having to go through each of the [patterns].*

We also found that in their normal live coding practice, several participants (P1, P2, P4, P6) reported that they regularly grouped multiple pattern changes together into a single block using Haskell's do notation. Essentially, this combines together several patterns into a single unit. This shows that live coders sometimes think about the evolution of a group of patterns together, which should be reflected in a versioning system.

6.5.2 ***Execution Sequence and Timing.*** Keeping a history of code executions with timing can allow a piece to be reconstructed exactly as it was. In other words, it enables *high-fidelity replay*. Both P2 and P6 saw value in this type of code replay for sharing their musical ideas with others (see 6.3.2) or for implementing an automatic replay feature for live performance (section 6.2.5).

During composition, P6 imagined being able to access the code execution history via a *global timeline* interface. Notably, taking the entire program state back in time would also solve the *cotemporal patterns* issue.

Not all participants thought the execution sequence and timing were important to track. P5 did not mind a more low-fidelity version capture, and felt that a piece was defined more by the logical patterns than its exact execution order.

> *(P5) The patterns exist a bit outside of time, you know. And so the order in which you decide to execute them, it's information on another level. I think it's not necessarily an impediment to not record them. I mean, for me, I'd say it's more something of the particular execution of a performance and not a property of the piece itself.*

7 DISCUSSION

Our findings reveal a set of design considerations for VCS centered on live coding music. As a medium, live coding music is situated at the intersection of performing arts (e.g., dance, music, theater), where artwork is embodied over time and the artifact is often ephemeral, and generative art, where code serves as the primary mechanism for creating artwork. As such, we discuss how our results can inform the design of VCS not only in the context of live coding music, but also in the larger contexts of the performing arts, generative arts, and beyond.

7.1 Expanding Musical Expressivity in Live Coding Music Performance

Our results suggest that SHARP expands the expressive capabilities of live coding musicians. We saw that SHARP provided new exploratory affordances through the ability to easily mix and match versions of different patterns (6.2.4). We also saw that SHARP facilitated incorporating musical form in performances on the fly (6.2.2). Both of these strategies represent lightweight ways to make potentially large sonic changes, partially addressing a known difficulty of the live coding medium [6, 49].

Although SHARP can facilitate big musical changes, it only helps when the new program state is a combination of already-seen versions. That is, creating completely new, unexplored patterns can still be challenging. Generative AI presents a possible opportunity for quickly making big code changes that explore new creative ideas in the conceptual space [5]. Past research suggests that generative artists may be amenable to using AI for larger creative leaps [1]. Incorporating generative AI in live coding music is an active area of research [39, 68, 69], and future work might explore how VCS can support human-AI interaction. For example, a human and AI agent might collaboratively write code snippets in a live coding performance, and a system like SHARP might augment version nodes with provenance to aid in sensemaking about the code history.

7.2 Versioning for Performing Arts

From our study, we found complexity in how version histories should be organized and presented to the user. SHARP organizes versions by what we might call *exploration order*, which does not capture the exact temporal relationship between versions (see 6.5). In particular, during performance, SHARP does not capture the exact *live performance order* of version executions; and during composition (i.e., not live), SHARP does not capture the *intended performance order* of version executions. Our participants were able to easily use SHARP's versions organized by exploration order, but still imagined a use for the live and intended performance order (e.g., for sharing or reviewing performances). In the context of other performance art practices, a VCS would similarly have to consider how to present versions based on the exploration order, live or intended performance order, or some combination thereof.

More fundamental is the question of what kind of versions are captured in the first place. For live coding, a VCS can record the history with varying degrees of fidelity: the performance instance (e.g., audiovisual recording), low-level interactions (e.g., keystroke recording for text replay [36]), execution/deployment history (e.g., tree graphs in SHARP), or a snapshot of algorithms (e.g., a file with user annotation). While SHARP captures an incomplete picture of a live coding performance, it also seems to create room for live coders' exploration (e.g., serendipitous discovery in 6.2.4). Previous work suggests that not being able to (fully) capture ideas can be a strategy to facilitate creativity [48, 62], and it remains an open question what the appropriate level of version capture is for different performing arts practices.

Complicating matters, some performance practices may have limited options for digital capture. While practices like traditional music composition might have several common digital representations (MIDI, MusicXML), practices like contemporary dance lack an equivalent standard [25]. Therefore, an additional challenge for designing a VCS for performing arts may be establishing a standard recorded notation to use in the system. Knotation is an illustrative example where researchers did not necessarily invent a new notation for choreography, but imported lo-fi documentation techniques (sketching, textual annotation) and augmented them with technical affordances (animation, multimodal presentation) in an interactive system [11].

7.3 Versioning for Generative Arts

We briefly discuss SHARP as it relates to two recent VCS designed for generative arts: Quickpose [55] and Spellburst [1]. In contrast to Quickpose and Spellburst, which both require the user to explicitly create new versions [1, 55], SHARP automatically creates a version for every code execution. We consider proactive, automatic versioning to be less interruptive, and thus essential for a live performance setting. Furthermore, as some participants suggested, automatic versioning can help artists remain in flow. On the other hand, our participants found the automatically created version trees in SHARP could get messy, often containing several superfluous versions. This is less likely to happen with Quickpose or Spellburst, as users can edit, organize, and prune version nodes in those systems [1, 55]. In fact, the ability to manipulate, group, and annotate the version graph in Quickpose was considered an essential design feature for embodying the creative process [55]. A future VCS might explore a hybrid approach where users can switch between automatic versioning for uninterrupted exploration and user-initiated management of the version graph to scaffold reflection.

7.4 Beyond Live Coding Music

Our findings may be applied to areas beyond music and the creative arts. One value of SHARP that participants identified is the ability to maintain clean code by eliminating ad-hoc versioning techniques like duplicating code. Programmers might like to use a lightweight versioning system like SHARP for more casual programming tasks, like creating a script for personal use. This could help them understand their own process and may make cleaning up the code easier, which is important if they want to share their work with others [18, 27, 60]. If they want to share their version control graphs, too, they may appreciate editing techniques like pruning and merging nodes, as suggested by our participants.

Another domain that might benefit from the design of SHARP is CS education. Live programming, where an instructor writes code from scratch as part of a lecture, is considered to be an effective pedagogical strategy [53, 59]. In a way, it is similar to live coding music: the instructor *performs* in front of the *audience* (students), with the flexibility of improvisation (e.g., debugging; revising in response to a student's question). Just as our participants felt SHARP's version trees might help the audience more easily follow their performance, perhaps a similar VCS could help students follow an instructor's live programming lecture. Post-lecture, sharing the live coding demonstration with students asynchronously in the form of a version tree may be more effective than the current method of sharing the final code text file or a screen recording. As another possibility, a collaborative VCS could sync the instructor's changes to each student's computer, allowing students to create private branches to explore the code on their own while easily being able to revert back to the instructor's version. In this case, perhaps the instructor could visualize students' code evolution, giving them insight into their class's performance. Recent research has demonstrated that such graphical visualization at scale can help instructors identify common issues and mistakes among students [74]. We plan to further explore the potential of live coding VCS to enhance the effectiveness of live programming in an educational setting.

Lastly, modern content in emerging social media emphasizes the performative aspect of creators and the liveness of creative practice. Such content includes live creative streaming [20], cooking videos [46], writing environments [9], and how-to videos [64]. This trend can also apply to interactive and collaborative systems [9, 33]. Designing tools to support the ideation, planning, and production of such content can benefit from exploration and improvisation support facilitated by VCS like SHARP.

8 CONCLUSION

In this study, we explored the enhancement of live coding music practices using SHARP and identified the specific needs of VCS in this context. We discovered that SHARP effectively supported live coding musicians' preparation and performances, fostering exploration and enabling the creation of novel musical aesthetics that were previously challenging to achieve in live coding. Additionally, we showcased multifaceted perspectives of version control, particularly those unique to the realms of performance and computational art. We also discussed how versioning is contingent on its contextual use. Our findings contribute to a deeper understanding of design implications associated with VCS in creative practices by examining the integration of VCS into live coding practices.

ACKNOWLEDGMENTS

We would like to thank our study participants for the time and energy they put into the study and for their thoughtful reflections and discussions. We would also like to thank the Tidal Cycles community for all their great work building, maintaining, and documenting the language.

REFERENCES

[1] Tyler Angert, Miroslav Suzara, Jenny Han, Christopher Pondoc, and Hariharan Subramonyam. 2023. Spellburst: A Node-Based Interface for Exploratory Creative Coding with Natural Language Prompts. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 100, 22 pages. https://doi.org/10.1145/3586183.3606719

[2] Mary Beth Kery and Brad A. Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 25–29. https://doi.org/10.1109/VLHCC.2017.8103446

[3] Mary Beth Kery and Brad A. Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 25–29. https://doi.org/10.1109/VLHCC.2017.8103446 ISSN: 1943-6106.

[4] Alan F Blackwell, Emma Cocker, Geoff Cox, Alex McLean, and Thor Magnusson. 2022. *Live coding: a user's manual.* MIT Press.

[5] Margaret A Boden. 2004. *The creative mind: Myths and mechanisms.* Routledge.

[6] Andrew R. Brown and Andrew C. Sorensen. 2007. aa-cell in practice: an approach to musical live coding. In *International Computer Music Conference*. 292–299. http://www.computermusic.org/

[7] Karen Burland and Alex McLean. 2016. Understanding live coding events. *International Journal of Performance Arts and Digital Media* 12, 2 (2016), 139–151. https://doi.org/10.1080/14794713.2016.1227596 arXiv:https://doi.org/10.1080/14794713.2016.1227596

[8] Karen Burland and Alex McLean. 2016. Understanding live coding events. *International Journal of Performance Arts and Digital Media* 12, 2 (2016), 139–151.

[9] Dashiel Carrera and Sang Won Lee. 2022. Watch Me Write: Exploring the Effects of Revealing Creative Writing Process through Writing Replay. In *Proceedings of the 14th Conference on Creativity and Cognition* (Venice, Italy) *(C&C '22)*. Association for Computing Machinery, New York, NY, USA, 146–160. https://doi.org/10.1145/3527927.3532806

[10] Kathy Cheng, Phil Cuvin, Alison Olechowski, and Shurui Zhou. 2023. User Perspectives on Branching in Computer-Aided Design. *Proc. ACM Hum.-Comput. Interact.* 7, CSCW2, Article 371 (oct 2023), 30 pages. https://doi.org/10.1145/3610220

[11] Marianela Ciolfi Felice, Sarah Fdili Alaoui, and Wendy E. Mackay. 2018. Knotation: Exploring and Documenting Choreographic Processes. In *Proceedings of the 2018*

CHI Conference on Human Factors in Computing Systems (Montreal, QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3174022

[12] Victoria Clarke, Virginia Braun, and Nikki Hayfield. 2015. Thematic analysis. *Qualitative psychology: A practical guide to research methods* 3 (2015), 222–248.

[13] Nick Collins. 2003. Generative music and laptop performance. *Contemporary Music Review* 22, 4 (2003), 67–79.

[14] Nick Collins. 2011. Live Coding of Consequence. *Leonardo* 44, 3 (June 2011), 207–211. https://doi.org/10.1162/LEON_a_00164

[15] Nick Collins, Alex McLEAN, Julian Rohrhuber, and Adrian Ward. 2003. Live coding in laptop performance. *Organised Sound* 8, 3 (Dec. 2003), 321–330. https://doi.org/10.1017/S135577180300030X

[16] Jerome W Crowder, Jonathan S Marion, and Michele Reilly. 2015. File naming in digital media research: Examples from the humanities and social sciences. *Journal of Librarianship and Scholarly Communication* 3, 3 (2015).

[17] Georgios Diapoulis. 2023. Musical Live Coding in Relation to Interactivity Variations. *Organised Sound* 28, 2 (2023), 149–161. https://doi.org/10.1017/S1355771823000444

[18] Helen Dong, Shurui Zhou, Jin L.C. Guo, and Christian Kästner. 2021. Splitting, Renaming, Removing: A Study of Common Cleaning Activities in Jupyter Notebooks. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. 114–119. https://doi.org/10.1109/ASEW52652.2021.00032

[19] Jules Françoise, Sarah Fdili Alaoui, and Yves Candau. 2022. CO/DA: Live-Coding Movement-Sound Interactions for Dance Improvisation. In *CHI Conference on Human Factors in Computing Systems*. ACM, New Orleans LA USA, 1–13. https://doi.org/10.1145/3491102.3501916

[20] C. Ailie Fraser, Joy O. Kim, Alison Thornsberry, Scott Klemmer, and Mira Dontcheva. 2019. Sharing the Studio: How Creative Livestreaming can Inspire, Educate, and Engage. In *Proceedings of the 2019 Conference on Creativity and Cognition* (San Diego, CA, USA) *(C&C '19)*. Association for Computing Machinery, New York, NY, USA, 144–155. https://doi.org/10.1145/3325480.3325485

[21] Andrew Goldman. 2019. Live coding helps to distinguish between embodied and propositional improvisation. *Journal of New Music Research* 48, 3 (2019), 281–293.

[22] Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. 2008. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (Monterey, CA, USA) *(UIST '08)*. Association for Computing Machinery, New York, NY, USA, 91–100. https://doi.org/10.1145/1449715.1449732

[23] Lile Hattori, Marco D'Ambros, Michele Lanza, and Mircea Lungu. 2011. Software Evolution Comprehension: Replay to the Rescue. In *2011 IEEE 19th International Conference on Program Comprehension*. 161–170. https://doi.org/10.1109/ICPC.2011.39

[24] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. 2008. Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. *IEEE Transactions on Visualization and Computer Graphics* 14, 6 (2008), 1189–1196. https://doi.org/10.1109/TVCG.2008.137

[25] Anna Heyward. 2015. How to Write a Dance: Remy Charlip and the problems of dance notation. *The Paris Review* (2015).

[26] Hilary Hutchinson, Wendy Mackay, Bo Westerlund, Benjamin B. Bederson, Allison Druin, Catherine Plaisant, Michel Beaudouin-Lafon, Stéphane Conversy, Helen Evans, Heiko Hansen, Nicolas Roussel, and Björn Eiderbäck. 2003. Technology Probes: Inspiring Design for and with Families. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Ft. Lauderdale, Florida, USA) *(CHI '03)*. Association for Computing Machinery, New York, NY, USA, 17–24. https://doi.org/10.1145/642611.642616

[27] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1265–1276. https://doi.org/10.1145/3025453.3025626

[28] Mary Beth Kery, Bonnie E. John, Patrick O'Flaherty, Amber Horvath, and Brad A. Myers. 2019. Towards Effective Foraging by Data Scientists to Find Past Analysis Choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, 1–13. https://doi.org/10.1145/3290605.3300322

[29] Mary Beth Kery and Brad A. Myers. 2018. Interactions for Untangling Messy History in a Computational Notebook. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 147–155. https://doi.org/10.1109/VLHCC.2018.8506576 ISSN: 1943-6106.

[30] Ryan Kirkbride. 2019. CodeBank: Exploring public and private working environments in collaborative live coding performance. In *International Conference on Live Coding (ICLC)* (Madrid, Spain).

[31] Scott R. Klemmer, Michael Thomsen, Ethan Phelps-Goodman, Robert Lee, and James A. Landay. 2002. Where do web sites come from? capturing and interacting with design history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Minneapolis, Minnesota, USA) *(CHI '02)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/503376.503378

[32] Byong G. Lee, Kai H. Chang, and N. Hari Narayanan. 1998. An integrated approach to version control management in computer supported collaborative writing. In *Proceedings of the 36th Annual Southeast Regional Conference (ACM-SE 36)*. Association for Computing Machinery, New York, NY, USA, 34–43. https://doi.org/10.1145/275295.275302

[33] Sang Won Lee. 2019. Liveness in Interactive Systems. *arXiv preprint arXiv:1910.02377* (2019).

[34] Sang Won Lee. 2019. Show Them My Screen: Mirroring a Laptop Screen as an Expressive and Communicative Means in Computer Music.. In *NIME*. 443–448.

[35] Sang Won Lee and Georg Essl. 2014. Communication, Control, and State Sharing in Networked Collaborative Live Coding. (2014).

[36] Sang Won Lee and Georg Essl. 2015. Live writing: Asynchronous playback of live coding and writing. In *Proceedings of the International Conference on Live Coding*.

[37] Jingyi Li, Sonia Hashim, and Jennifer Jacobs. 2021. What We Can Learn From Visual Artists About Software Development. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3411764.3445682

[38] Jon Loeliger and Matthew McCullough. 2012. *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, Inc.

[39] Norah Lorway, Edward Powley, and Arthur Wilson. 2021. Autopia: An AI collaborator for live networked computer music performance. In *Proceedings of the Second Conference on AI Music Creativity*.

[40] Thor Magnusson. 2011. Algorithms as Scores: Coding Live Music. *Leonardo Music Journal* 21 (12 2011), 19–23. https://doi.org/10.1162/LMJ_a_00056 arXiv:https://direct.mit.edu/lmj/article-pdf/doi/10.1162/LMJ_a_00056/1675039/lmj_a_00056.pdf

[41] Alex McLean. 2014. Making programming languages to dance to: live coding with tidal. In *Proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design*. 63–70.

[42] Alex McLean and Geraint Wiggins. 2010. Live Coding Towards Computational Creativity. In *Proceedings of the First International Conference on Computational Creativity*. https://doi.org/10.5281/zenodo.7219926

[43] Alex G. R. Mclean, Dave Griffiths, Nick Collins, and Geraint Wiggins. 2010. Visualisation of live code. (July 2010). https://doi.org/10.14236/ewic/EVA2010.6 Publisher: BCS Learning & Development.

[44] Hiroaki Mikami, Daisuke Sakamoto, and Takeo Igarashi. 2017. Micro-Versioning Tool to Support Experimentation in Exploratory Programming. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6208–6219. https://doi.org/10.1145/3025453.3025597

[45] Mathieu Nancel and Andy Cockburn. 2014. Causality: a conceptual model of interaction history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 1777–1786. https://doi.org/10.1145/2556288.2556990

[46] Megha Nawhal, Jacqueline B Lang, Greg Mori, and Parmit K Chilana. 2019. VideoWhiz: Non-Linear Interactive Overviews for Recipe Videos.. In *Graphics Interface*. 15–1.

[47] Molly Jane Nicholas, Sarah Sterman, and Eric Paulos. 2022. Creative and Motivational Strategies Used by Expert Creative Practitioners. In *Proceedings of the 14th Conference on Creativity and Cognition* (Venice, Italy) *(C&C '22)*. Association for Computing Machinery, New York, NY, USA, 323–335. https://doi.org/10.1145/3527927.3532870

[48] Molly Jane Nicholas, Sarah Sterman, and Eric Paulos. 2022. Creative and Motivational Strategies Used by Expert Creative Practitioners. In *Creativity and Cognition (C&C '22)*. Association for Computing Machinery, New York, NY, USA, 323–335. https://doi.org/10.1145/3527927.3532870

[49] Click Nilson. 2007. Live coding practice. In *Proceedings of the 7th international conference on New interfaces for musical expression - NIME '07*. ACM Press, New York, New York, 112. https://doi.org/10.1145/1279740.1279760

[50] David Ogborn, Jamie Beverley, Luis Navarro del Angel, Eldad Tsabary, Alex McLean, and Esteban Betancur. 2017. Estuary: Browser-based collaborative projectional live coding of musical patterns. In *International Conference on Live Coding (ICLC)*, Vol. 2017.

[51] Shaun Phillips, Jonathan Sillito, and Rob Walker. 2011. Branching and merging: an investigation into current version control practices. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering* (Waikiki, Honolulu, HI, USA) *(CHASE '11)*. Association for Computing Machinery, New York, NY, USA, 9–15. https://doi.org/10.1145/1984642.1984645

[52] Arrian Purcell, Henry Gardner, and Ben Swift. 2014. Visualising a live coding arts process. In *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: the Future of Design (OzCHI '14)*. Association for Computing Machinery, New York, NY, USA, 141–144. https://doi.org/10.1145/2686612.2686634

[53] Adalbert Gerald Soosai Raj, Jignesh M. Patel, Richard Halverson, and Erica Rosenfeld Halverson. 2018. Role of Live-coding in Learning Introductory Programming. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research (Koli Calling '18)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3279720.3279725

[54] N. Rama Rao and K. Chandra Sekharaiah. 2016. A Methodological Review Based Version Control System with Evolutionary Research for Software Processes. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies* (Udaipur, India) *(ICTCS '16)*. Association for Computing Machinery, New York, NY, USA, Article 14, 6 pages. https://doi.org/10.1145/2905055.2905072

[55] Eric Rawn, Jingyi Li, Eric Paulos, and Sarah E. Chasins. 2023. Understanding Version Control as Material Interaction with Quickpose. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–18. https://doi.org/10.1145/3544548.3581394

[56] Charlie Roberts and Graham Wakefield. 2018. Tensions & Techniques in Live Coding Performance. In *The Oxford Handbook of Algorithmic Music*. https://doi.org/10.1093/oxfordhb/9780190226992.013.20 Journal Abbreviation: The Oxford Handbook of Algorithmic Music.

[57] Charles Roberts, Matthew Wright, and JoAnn Kuchera-Morin. 2015. Beyond editing: extended interaction with textual code fragments.. In *NIME*. 126–131.

[58] Emilia Rosselli Del Turco and Peter Dalsgaard. 2023. "I wouldn't dare losing one": How music artists capture and manage ideas. In *Proceedings of the 15th Conference on Creativity and Cognition* (Virtual Event, USA) *(C&C '23)*. Association for Computing Machinery, New York, NY, USA, 88–102. https://doi.org/10.1145/3591196.3593338

[59] Marc J. Rubin. 2013. The effectiveness of live-coding to teach introductory programming. In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 651–656. https://doi.org/10.1145/2445196.2445388

[60] Adam Rule, Ian Drosos, Aurélien Tabard, and James D. Hollan. 2018. Aiding Collaborative Reuse of Computational Notebooks with Annotated Cell Folding. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 150 (nov 2018), 12 pages. https://doi.org/10.1145/3274419

[61] Ben Shneiderman. 2007. Creativity Support Tools: Accelerating Discovery and Innovation. *Commun. ACM* 50, 12 (dec 2007), 20–32. https://doi.org/10.1145/1323688.1323689

[62] Sarah Sterman, Molly Jane Nicholas, and Eric Paulos. 2022. Towards Creative Version Control. *Proc. ACM Hum.-Comput. Interact.* 6, CSCW2, Article 336 (nov 2022), 25 pages. https://doi.org/10.1145/3555756

[63] Ben Swift, Andrew Sorensen, Henry Gardner, and John Hosking. 2013. Visual code annotations for cyberphysical programming. In *2013 1st International Workshop on Live Programming (LIVE)*. IEEE, 27–30.

[64] Sonja Utz and Lara N. Wolfers. 2022. How-to videos on YouTube: the role of the instructor. *Information, Communication & Society* 25, 7 (2022), 959–974. https://doi.org/10.1080/1369118X.2020.1804984 arXiv:https://doi.org/10.1080/1369118X.2020.1804984

[65] Konstantinos VASİLAKOS. 2021. Exploring Live Coding as Performance Practice. *Porte Akademik Müzik ve Dans Araştırmaları Dergisi* 21 (2021), 21–37. https://doi.org/10.59446/porteakademik.1033868

[66] Ge Wang and Perry R. Cook. 2004. 2004: On-the-Fly Programming: Using Code as an Expressive Musical Instrument. In *A NIME Reader*, Alexander Refsum Jensenius and Michael J. Lyons (Eds.). Vol. 3. Springer International Publishing, Cham, 193–210. https://doi.org/10.1007/978-3-319-47214-0_13 Series Title: Current Research in Systematic Musicology.

[67] Nathaniel Weinman, Steven M. Drucker, Titus Barik, and Robert DeLine. 2021. Fork It: Supporting Stateful Alternatives in Computational Notebooks. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3411764.3445527

[68] Elizabeth Wilson, György Fazekas, and Geraint Wiggins. 2023. On the Integration of Machine Agents into Live Coding. *Organised Sound* 28, 2 (2023), 305–314. https://doi.org/10.1017/S1355771823000420

[69] E Wilson, S Lawson, A McLean, J Stewart, et al. 2021. Autonomous Creation of Musical Pattern from Types and Models in Live Coding. In *Proceedings of the Ninth Conference on Computation, Communication, Aesthetics & X*.

[70] Moritz Wittenhagen, Christian Cherek, and Jan Borchers. 2016. Chronicler: Interactive Exploration of Source Code History. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 3522–3532. https://doi.org/10.1145/2858036.2858442

[71] YoungSeok Yoon and Brad A. Myers. 2015. Supporting Selective Undo in a Code Editor. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 223–233. https://doi.org/10.1109/ICSE.2015.43 ISSN: 1558-1225.

[72] YoungSeok Yoon, Brad A. Myers, and Sebon Koo. 2013. Visualization of fine-grained code change history. In *2013 IEEE Symposium on Visual Languages and Human Centric Computing*. 119–126. https://doi.org/10.1109/VLHCC.2013.6645254

[73] Young Seok Yoon and Brad A. Myers. 2014. A longitudinal study of programmers' backtracking. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 101–108. https://doi.org/10.1109/VLHCC.2014.6883030

[74] Ashley Ge Zhang, Yan Chen, and Steve Oney. 2023. VizProg: Identifying Misunderstandings By Visualizing Students' Coding Progress. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 596, 16 pages. https://doi.org/10.1145/3544548.3581516

[75] Lei Zhang, Ashutosh Agrawal, Steve Oney, and Anhong Guo. 2023. VRGit: A Version Control System for Collaborative Content Creation in Virtual Reality. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 36, 14 pages. https://doi.org/10.1145/3544548.3581136

[76] Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. 2018. Version Control System: A Review. *Procedia Computer Science* 135 (2018), 408–415. https://doi.org/10.1016/j.procs.2018.08.191 The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life.

[77] Weiqin Zou, Weiqiang Zhang, Xin Xia, Reid Holmes, and Zhenyu Chen. 2019. Branch Use in Practice: A Large-Scale Empirical Study of 2,923 Projects on GitHub. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. 306–317. https://doi.org/10.1109/QRS.2019.00047

A APPENDIX

A.1 Diary Study Prompt

Enter your diary entry below. You can include anything you want, but here are some guiding questions:

- How did you use SHARP during the session?
- How did it help (or hinder) you?
- How did it influence what you created musically?
- How did you start your session? How did you wrap up?

A.2 Exit Interview Guiding Questions

- How did you find using SHARP?
- How did it change how you live coded?
- Did it feel freeing? Constraining?
- Did you feel it influenced the music you ended up making?
- Did it make things faster, slower, easier or harder?
- Did you find it better for practice/exploration or performance?
- What features did you use/not use?
- What features would you have wanted?
- How true to a normal performance did your recording of the performance feel?
- What does the ideal versioning system for live coding look like? What features would it have?
- Would you want to use something like SHARP going forward?