Herding cats: observing live coding in the wild

Article (Published Version)

Thor Magnusson
Department of Music
University of Sussex
Falmer, Brighton, BN1 9QJ, UK
T.Magnusson@sussex.ac.uk

Herding Cats: Observing Live Coding in the Wild

Abstract: After an eventful decade of live-coding activities, this article seeks to explore the practice with the aim of situating it in the history of contemporary arts and music. The article introduces several key points of investigation in live-coding research and discusses some examples of how live-coding practitioners engage with these points in their system design and performances. In light of the extremely diverse manifestations of live-coding activities, the problem of defining the practice is discussed, and the question is raised whether live coding is actually necessary as an independent category.

This journal issue celebrates the tenth anniversary of organized live coding (see www.toplap.org). Numerous live-coding systems, festivals, conference tracks, journal issues, research projects, and club nights have appeared and introduced the practice to diverse fields of art, music, and science (in particular, computer science). As an arts practice, it has its roots in musical performance, but live coding has become common in visual arts, light systems, robotics, dance, poetry, and other art forms that operate with algorithmic instructions. In the context of this journal issue, it seems appropriate to stress the origin of live-coding practice in the arts, without failing to mention that the related term of "live programming" has been used for a considerably longer time in certain research tracks of computer science, particularly in the field of programming-language design. Researchers like Ungar and Smith explored live-programming practice as early as 1969 (Ungar and Smith 2013) and, in an article on the visual programming language VIVA, Tanimoto (1990) defined "liveness" as an attribute of programming. Live programming is thus perceived to be a useful method in activities ranging from re-programming factory production lines (Swift et al. 2013) to making music in nightclubs. This article will describe live coding as a unique practice in a strong relationship with live programming but stress its origins in live performance.

Journal articles and conference papers on live coding, typically written by the protagonists themselves, have introduced the practice (Collins et al. 2003; Ward et al. 2004; McLean 2008), explored it in a computer science context (Blackwell and Collins 2005; Rohrhuber, de Campo, and Wieser 2005; McLean and Wiggins 2010; Sorensen and Gardner 2010), described particular systems and solutions (Sorensen 2005; Wakefield, Smith, and Roberts 2010; Freeman and Van Troyer 2011; Magnusson 2011a; McLean and Wiggins 2011; Roberts, Wakefield, and Wright 2013), explored live coding as musical scores (Blackwell and Collins 2005; Magnusson 2011b), and contextualized it as an embodied musical practice that requires practicing on a par with acoustic instruments (Sorensen and Brown 2007; Aaron et al. 2011; Collins 2011). After a decade of fruitful experiments, it is perhaps time to reflect on how live coding has operated within the performing arts. Discussing a selection of highly diverse live-coding systems, this article seeks to establish what they may have in common, resulting in a discussion of the problems of defining live coding, and, in the process, to introduce weak and strong criteria for the practice.

Computer Music Journal, 38:1, pp. 8–16, Spring 2014
doi:10.1162/COMJ_a_00216

On Naturalizing Live Coding

Live coding does not have a particular unified aesthetic in terms of musical or visual style. Nevertheless, the practice has at times been perceived as a movement, akin to movements found in 20th century modernism. We find a strong emphasis on formal experiments, reductionism, and functionalism. There are manifestos, key texts, and custom coding platforms. A visit to a nightclub hosting a live-coding event might even conjure up images of exclusive avant-garde practices, where live coders perform for other coders already initiated into the wicked "sourcery" of programming computers. This art of writing algorithms for binary machines can be so alien and obscure to the audience that the situation almost recalls the difficulty people had understanding postwar European avant-garde music.

This image is as familiar as it is misleading. It is true that the birth of live coding draws from modernist practices (with its manifestos, rules, and imperatives) but this is inevitable, as formalism is—and formal experiments are—a necessary aspect in the exploration of a new medium; suffice it to mention video art experiments by Nam June Paik and the Vasulkas in the 1960s, or net.art work by artists such as Alexei Shulgin and Jodi in the late 1990s. Such a formalism involves a deep exploration of the properties of the medium at hand, and we find an analogy in the way live coders have designed and performed with their systems. In live coding, however, the investigation tends to be a formalism of thought and the language or system of encoding it, as opposed to artistic content, which may appear formalist or not. Perhaps the situation is better described by the modernist critic Clement Greenberg, who defines it as one in which the two aspects become inseparable:

> Content is to be dissolved so completely into form that the work of art or literature cannot be reduced in whole or in part to anything not itself [. . .] In turning his attention away from subject matter of common experience, the poet or artist turns it in upon the medium of his own craft (Greenberg 1961, p. 6).

The performance-art elements of live coding differentiate it from the pure self-referentiality of formalist modernism, however—although there could, of course, exist a purely conceptual live coding without any other output than the code itself. I am not aware of such an approach in the area of live coding, although it exists in off-line coding, perhaps best exemplified by Pall Thayer's *Microcodes* (Myers 2009). The live-coding activities that resemble the aforementioned modernist tendencies can be explained with a common trajectory that takes place when a new artistic format develops. The initial focus is on the formal part of the practice, on the medium or the tool, and it is only later, when the technology undergoes a process of "naturalization," that the focus shifts elsewhere:

> The more naturalized the object becomes, the more unquestioning the relationship of the community to it; the more invisible the contingent and historical circumstances of its birth, the more it sinks into the community's routinely forgotten memory (Bowker and Star 2000, p. 299).

Live-coding practice has consciously put effort into expediting this naturalization process. It deliberately engages with the audience through various channels, such as sitting among them while performing (as with the band PowerBooks_UnPlugged), allowing people to contribute to the coding of dance performers (e.g., Kate Sicchio's work), or submitting code through Twitter (as in my own work with the live-coding environment "ixi lang"). One of the fundamental tenets of the TOPLAP manifesto (available online at toplap.org/wiki/ManifestoDraft) is "Show us your screens." It is an explicit act of audience inclusion, responding to the common laptop-performance format where relatively simple interfaces are used but not shown, even though many are commonly known to the audience. This is taking the etymology of the word "program" seriously, as the Greek root, *prográphein*, signifies the activity of public writing (Hoad 1996). Live coders have also been prolific in explaining their practice and systems of writing code, with journal articles, online discussions, and similar activities (this article [and this paragraph] being an example of such recursion).

Live Coding in the Wild

Live coding is a heterogeneous practice and thus somewhat hard to define. It involves a multiplicity of approaches that have one thing in common: Algorithmic instructions are written in real time. Collins (2011, p. 209) states that "the more profound the live coding, the more a performer must confront the running algorithm, and the more significant the intervention in the works, the deeper the coding act." For Collins, most performances fail to "live up to this promise." The argument, also expressed by McLean (2008), is that a true "liveness" requires that the performer is not simply manipulating pre-written code in real time, but is actually writing and

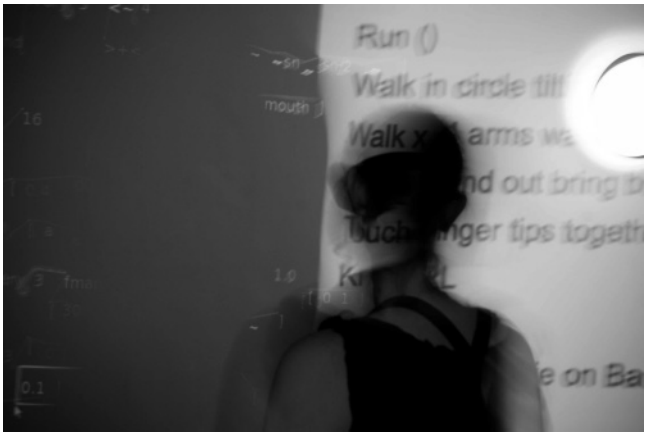


Figure 1. Kate Sicchio dancing with code. (Photograph by Bill Best, Sheffield, UK, Creative Commons BY-NC-ND 3.0.)

modifying algorithms during the execution of the program. This is arguably a prerequisite of live coding, as simply running code and changing parameters in prewritten code is more comparable to operating buttons, sliders, and knobs on a screen interface or a MIDI controller. We could talk about weak and strong definitions of live coding, with the weak one allowing trivial code manipulations and strong one adhering to Collins's and McLean's position. The strong definition would result in the conclusion that some so-called live-coding performances do not really include any live coding at all.

In the absence of any clear definitions of live coding, apart from the requirement that algorithms are written in real time, it may be appropriate to take a sample of some prolific live coders and their systems, and ask whether we can detect an aesthetic style, or any specific performance contexts or technological foundations. The examples here are chosen with the aim of demonstrating the variety of the practice, leaving out other significant contributors to live coding whose performance setup might be similar. The aim is to study how live coding operates in a live context, and to outline the problems practitioners are trying to address in their work.

PowerBooks_UnPlugged is an ensemble of six performers who make use of wireless networks to communicate and share code in a networked live-coding performance. As the name indicates, the laptops are unplugged, and the only sound output is from the built-in laptop speakers. Members of the collective sit among the audience, blurring the traditional performer–audience spatial divide, while sending code snippets to each other for collaborative coding (Rohrhuber et al. 2007). The ensemble describes itself as "the first acoustic computer music folk band" (for instance, on the band Web site at pbup.goto10.org) with a reference to the demystification of the laptop as a musical instrument.

One of the early live-coding collectives is a group called Slub, consisting of three members who perform with different live-coding systems of their own creation and who collaborate over a network using a shared musical time clock. Their systems are well known for their experimental and innovative aspects, focusing on language design and human–computer interaction. Slub regularly uses multiple projectors to overlay the performers' desktop images, an act of obfuscation where the rendered image on the wall becomes a visual montage of an aesthetic dimension. The collective often collaborates with other artists, recently with performance artist Hester Reeves.

Nick Collins has explored live coding in various forms and projects. Together with Frederik Olofsson, he once set aside a month to practice live coding on a daily basis, exploring the topics of practice and coding virtuosity. This was documented in a 2007 paper (Nilson 2007). Collins, sometimes encrypted as "Click Nilson" (although rumor has it that Nilson is retired), has written work for the live coding of human improvising musicians (Collins 2011), composed live coding scores (Collins 2012), and engaged in duels with other live-coding heavyweights such as Alex McLean and Ge Wang (Nilson 2007), as well as with the master of live electronics and hardware hacking, Nic Collins (not to be confused with Nick Collins).

Kate Sicchio (see Figure 1) is a choreographer and dancer who uses live coding in the context of dance. In her performances, Sicchio works with algorithmic choreographic instructions, a kinetographic encoding of human movements, which can be explored in a real-time performance. Sicchio finds that, unlike other choreographic languages, textual programming can provide an expressive syntax for describing algorithms that afford generative interpretations for human interpreters (www.sicchio.com).

Figure 2. Wrongheaded preaching code at the Arnolfini in Bristol. (Photograph by Megan Farrow, Bristol.)

Figure 3. Benoît and the Mandelbrots playing their greatest hits. (Photograph by Daniel Bollinger, www.danielbollinger.de.)

Dave Griffiths, a member of aforementioned collective Slub, has created various live-coding systems, notably the Scheme-based Fluxus for graphical live coding, Al-Jazari (where he uses a gamepad to program robots that make music), and Scheme Bricks (a graphical block-type interface for programming music and visuals). Griffiths's work tends to contain a strong visual element, inspired by computer games and agent-based programming (Griffiths 2007), and is thus ideally suited as a first step in coding for novices and children.

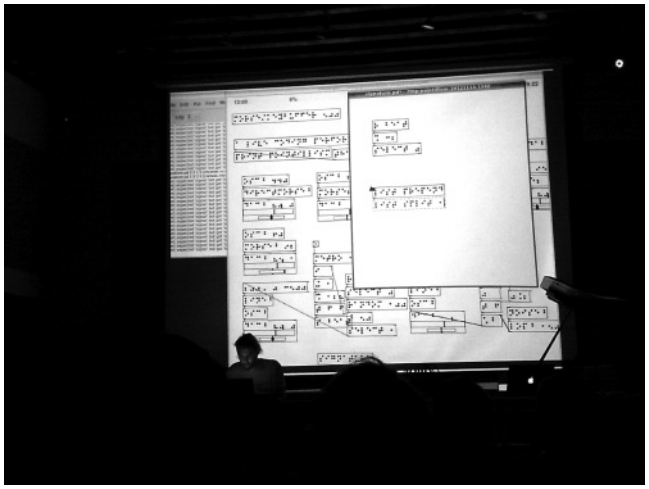
The group Wrongheaded (see Figure 2) is a collaboration between live coders Click Nilson and Matt Yee-King. Their performances include strong performative aspects and are typically humorous, sometimes to the degree that the music is all but forgotten among tactics that might involve expressive dance, the autopsy of teddy bears, or a religious exegesis of existential algorithms. This takes place concurrently with the writing of SuperCollider synthesizer definitions and the configuration of sound card drivers for the Linux-based Raspberry Pi.

There is also a boy-band dimension in live coding, most notably present in the collective Benoît and the Mandelbrots (see Figure 3). This multilingual (both in terms of human and machine languages) supergroup continually surprises with their advanced systems, coding expertise, and solutions to the problems of networked computer music and collaborative audiovisual performance.

Andrew Sorensen and Andrew Brown perform under the name aa-cell (Sorensen and Brown 2007). They use a Scheme-based environment called Impromptu, written by Sorensen (2005). Their performances have impressed audiences around the world through their range of expression, coding eloquence, and compositional structures. If one were to name virtuoso live coders, algorithmic equivalents to Paganini and Liszt, the names of these performers are bound to be included. Sorensen is currently working on a new live-coding environment called Extempore for "cyber-physical programming" (Sorensen and Gardner 2010).

Jason Freeman and Akito Van Troyer (2011) have created a live-coding environment called LOLC.

Figure 4. IOhannes m zmölnig live coding with Pure Data. (Photograph by David Griffiths, Falmouth, UK.)



This simulates a chat client and allows novices in computer programming to live code music in larger ensembles over a wireless network in the same location or in a distributed performance. The system encourages not only conversation and collaboration between performers, but also the borrowing and adoption of code, practically rendering the question of authorship in this collaborative performance setup meaningless. A more recent system called SGLC (Lee and Freeman 2013) builds on LOLC, but aims at the inclusion of acoustic instruments through the use of traditional staff notation.

Mark Havryliv uses his *P[a]ra[pra]xis* live-coding software to sonify text (typically poetry). Words are treated as objects with properties and relationships to other words—i.e., the software is not a one-to-one mapping of characters or letters to sound, but focuses on linguistic conditions. After a word is typed, the system can change it to another semantically or syntactically related word. Knowing the system well, it is possible to code complex musical and linguistic patterns (Dubrau and Havryliv 2010).

Most of the these performance systems are text-based, but graphical data-flow programming environments like Pure Data (Pd) and Max/MSP also allow for live coding. IOhannes m zmölnig (see Figure 4), for instance, is a Pd developer who has created a live-coding system for multiple users and used it extensively (zmölnig 2007). He is also known for his "pointillistic" live coding of Morse code with a Braille font.

Gibber (Roberts and Kuchera-Morin 2012) is a system that takes live coding into the realm of the Web browser. Using the new Web Audio technologies for JavaScript, the system enables people to visit a Web page and start creating synthesizers or composing music, without needing to download and install any additional software. Multiple users can contribute in the same session in a networked performance manner.

Till Bovermann and Sara Hildebrand Marques Lopes have devised a piece called Oulipop, where they perform "meta-level" live coding by manipulating text according to the algorithmic rules specified by the *Ouvroir de literature potentielle* ("workshop for potential literature," abbreviated as Oulipo). The Oulipop system runs a low-level synthesis engine that translates ASCII characters into assembler instructions for a virtual chip, the operation of which is sonified. The piece is performed by Bovermann and Hildebrand Marques Lopes on stage, where they interfere with each other's text, thus turning what might have been meaningful prose into good-sounding nonsense (Bovermann and Griffiths 2014).

Craig Latta's system, Quoth, implements a musical performance system in the form of a text adventure game command line interface. Quoth is a natural language processor, presented in the form of interactive fiction, that treats the English language as executable code (an example is available online at vimeo.com/50530082).

In my own work I have developed two live-coding systems, the ixi lang (Magnusson 2011a) and the Threnoscope (Magnusson 2013). Both of these systems are made with the aim of providing scope for musical expression that is both high level and constrained. The musical output of the two systems is distinctively different, with the former typically resulting in beat-orientated and melodic music (a strong focus is put on scales, chords, and harmony), whereas the latter practically removes time and emphasizes notes of indefinite length (drones) whose parameters can be controlled through a textual interface.

Loud Objects, a collaboration between Kunal Gupta, Tristan Perich, and Katie Shima, focuses

on creating hardware in a live, performative context. An overhead projector displays how members solder custom audio circuits using microchips, capacitors, and transducers. Although their performances consist solely of soldering hardware, they represent a form of live programming, because they involve the hot-wiring of control structures in real time.

The list of live-coding practitioners enumerated here could be much longer, but it sufficiently demonstrates that there are no specific tools, practices, or musical aesthetics at play: Live coding is a fuzzy concept representing a performance technique with a multitude of practices that do not share any one essential requirement, except perhaps that algorithms are written and operated on in real time. Each of the named practices addresses specific problems involved in composing with algorithms, some of which could be summarized as follows:

- Languages for algorithmic thinking
- Comprehensibility and intuitiveness of code
- Audience engagement and involvement
- Graphical visualizations of algorithms
- Virtuosity and the system's provision for speed coding
- Compositional expressiveness
- Direct access to the artistic material (e.g., synthesis or temporal patterns)
- Collaboration through network protocols
- "Liveness" and the ability to control existing structures
- Embodiment and physicality
- Live-coding systems as musical pieces or works of art

As a novel research field with a strong practical underpinning, live coding engages with diverse unexplored and novel areas in (musical) composition, human–computer interaction, programming-language design, and performance studies, rendering the practice an ideal platform for both artistic and scientific experiments. The examples in this article illustrate how live coding tries to communicate algorithmic thinking, real-time composition, and networked collaborations to the audience by designing innovative systems or exploring novel performance contexts. These examples typify approaches to humanizing machines by creating conversational interfaces with them, enabling artists to issue commands through the notation of code. The examples also demonstrate how live-coding systems can range from being open, general programming environments aimed at general users to being systems with much narrower scope, which can be seen as musical works in their own right. Boverman and Hildebrand Marques Lopes' Oulipop and my own Threnoscope exemplify such approaches.

Disparate Origins and Fuzzy Ends

Live coding has been explored in this article primarily as an art-based practice, but we shouldn't forget that while live coding has one foot in the arts, the other stands firmly in computer science. Having established itself in the past decade, live coding owes much to diverse foundational work in computer science since the 1950s. The coding languages vary, but interpreted programming languages are typically used, often as descendants of either Lisp- or Smalltalk-based systems, whose functional and object-oriented paradigms, respectively, have laid foundations to both textual and graphical live-coding environments. Just-in-time compilation (Aycock 2003) has been an influential technique supporting liveness in coding, for example in the JITLib approach implemented in SuperCollider by Rohrhuber in 2000 (Rohrhuber, de Campo, and Wieser 2005) or by Sorensen (2005) with his work on the Impromptu and Extempore environments. Visual or data-flow programming languages have also been inspiring, including Self (Smith and Ungar 1995) and Scratch (Resnick et al. 2009), an approach represented by Max/MSP and Pure Data in the field of media arts (Puckette 1988). Research into human–computer interaction, computer games, and new media interface design have also been important references in more experimental programming frameworks, for example, in the work of Griffiths (2007), McLean and Wiggins (2011), Latta's Quoth, and in ixi lang (Magnusson 2011a). We also find that the networked collaboration characteristic of many live-coding performances owes

much to the research and development of historic communication protocols, but also to newer community networks for collaborative coding, such as the GitHub host for the Git revision control system.

Live coding is the writing and performance, in real time, of music or other art forms, including games, where algorithms are the primary form of notation; it involves designing step-by-step rules for machines, humans, or others (e.g., animals or nature) to execute. An effective performance technique, live coding addresses the problems of improvisation and on-the-fly decision-making in live performance with machines. The fact that the machine can be redefined in real time opens up new avenues in compositional and musical performance practices, and, indeed, it seems to be a logical and necessary step in the evolution of human–machine communication. In the current technological condition, media formats need not be linear, deterministic, and static; their real-time rendering allows for interpretation, interaction, and change of the type we find in mobile apps, generative music, and games. Performance contexts will increasingly reflect the dynamic nature of modern media, where composers, performers, and audience (or any amalgamation thereof) are able to write or re-write notation, scores, or any other instructions and are able to design or re-design software, hardware, or other machinery in a more open and dynamic way.

Considering that live coding as a performance method represents a propitious and natural way of engaging with notation or instructions in real time, we might question whether the approach of defining live coding as a specific category is necessary from a longer-term perspective. At least we might rethink in which contexts it might be beneficial to maintain the category, because when the novelty wears off and the naturalization process has fully taken place, we may find the method blends so effortlessly into the diverse art forms that we don't need to talk about live coding anymore. In this future scenario, live coding simply becomes one of the most pertinent approaches among available performance techniques that allow for real-time composition and improvisation.

Conclusion

This article has discussed some manifestations of live coding and described how selected practitioners explore key research themes, while resisting the temptation to define live coding in greater detail than stating that it involves the writing of algorithms in real time. Attempts to define this wide field of activities more closely are likely to become an exercise in herding cats: There will always be examples that escape the constraints of such definitions. Additionally, the need for the category of live coding to be maintained as more than as a temporary tool has been questioned.

As a provisional instrument, however, the live-coding category serves an important purpose. The arrival of live coding in the historical timeline of contemporary music is important, and the tension between the strong and weak criteria for live coding happens to be strikingly analogous to the dichotomy between composition and performance in written sheet music since the 19th century. Historically, in the 17th and 18th centuries, musical scores were often seen as descriptive of music, rather than prescriptive; this is a distinction Lydia Goehr (1992, p. 188) frames as the difference between "composing through performance and composing prior to performance." From this perspective, live coding adopts a pre-Romantic method of composing through performance in real time, where everything remains open to change—the compositional process, the instrument design, and the intelligence of the system performing the piece.

Furthermore, live coding manages to blur most concepts of established musical discourse, such as composer, performer, and audience; instrument, score, and piece; composition, performance, and improvisation; stage and auditorium; and instrument and tool. It presents an interesting take on the modernist concern regarding form and content, and it blurs the concepts of presence and absence as well as notions of present, past, and future. In this respect, I think that live coding reaches much further back into musical history than what is clearly a practice with strong roots in modernism and computer science. It extends back to a time before these categories became

concrete and hard-wired definitions of musical practice.

Acknowledgments

I would like to thank *Computer Music Journal's* guest editor, Julian Rohrhuber, and Peter Castine for their invaluable input.

References

Aaron, S., et al. 2011. "A Principled Approach to Developing New Languages for Live Coding." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 381–386.

Aycock, J. 2003. "A Brief History of Just-in-Time." *ACM Computing Surveys* 35(2):97–113.

Blackwell, A., and N. Collins. 2005. "The Programming Language as a Musical Instrument." In *Proceedings of the Psychology of Programming Interest Group* (pages unnumbered).

Bovermann, T., and D. Griffiths. 2014. "Computation as Material in Live Coding." *Computer Music Journal* 38(1):40–53.

Bowker, G. C., and S. L. Star. 2000. *Sorting Things Out: Classification and Its Consequences*. Cambridge, Massachusetts: MIT Press.

Collins, N., et al. 2003. "Live Coding in Laptop Performance." *Organised Sound* 8(3):321–330.

Collins, N. 2011. "Live Coding of Consequence." *Leonardo* 44(3):207–211.

Collins, N. 2012. "Six Live Coding Works for Ensemble." Available online at www.sussex.ac.uk/Users/nc81/livecodingworksforensemble.html. Accessed July 2013.

Dubrau, J., and M. Havryliv. 2010. "P[a]ra[pra]xis: Towards Genuine Realtime 'Audiopoetry'." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 467–468.

Freeman, J., and A. Van Troyer. 2011. "Collaborative Textual Improvisation in a Laptop Ensemble." *Computer Music Journal* 35(2):8–21.

Goehr, L. 1992. *The Imaginary Museum of Musical Works: An Essay in the Philosophy of Music*. Oxford: Oxford University Press.

Greenberg, C. 1961. "Avant-Garde and Kitsch." In *Art and Culture: Critical Essays*. Boston, Massachusetts: Beacon Press, pp. 3–21.

Griffiths, D. 2007. "Game Pad Live Coding Performance." In J. Birringer, T. Dumke, and K. Nicolai, eds. *Die Welt als virtuelles Environment*. Dresden: Hellerau, pp. 169–179.

Hoad, T. F. 1996. *The Concise Oxford Dictionary of English Etymology*. Oxford: Oxford University Press.

Lee, S. W., and J. Freeman. 2013. "Real-Time Music Notation in Mixed Laptop-Acoustic Ensembles." *Computer Music Journal* 37(4):24–36.

Magnusson, T. 2011a. "The ixi lang: A SuperCollider Parasite for Live Coding." In *Proceedings of the International Computer Music Conference*, pp. 503–506.

Magnusson, T. 2011b. "Algorithms as Scores: Coding Live Music." *Leonardo Music Journal* 21(1):19–23.

Magnusson, T. 2013. "The Threnoscope: A Musical Work for Live Coding Performance." In *International Workshop on Live Programming at the International Conference on Software Engineering* (pages unnumbered).

McLean, A. 2008. "Live Coding for Free." In A. Mansoux and M. de Valk, eds. *Floss+Art*. London: OpenMute, pp. 224–231.

McLean, A., and G. Wiggins. 2010. "Bricolage Programming in the Creative Arts." In *Proceedings of the Psychology of Programming Interest Group* (pages unnumbered).

McLean, A., and G. Wiggins. 2011. "Texture: Visual Notation for the Live Coding of Pattern." In *Proceedings of the International Computer Music Conference*, pp. 621–628.

Myers, R. 2009. "Microcodes." *Furtherfield*. Available online at www.furtherfield.org/reviews/microcodes. Accessed July 2013.

Nilson, C. [N. Collins]. 2007. "Live Coding Practice." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 112–117.

Puckette, M. 1988. "The Patcher." In *Proceedings of the International Computer Music Conference*, pp. 420–429.

Resnick, M., et al. 2009. "Scratch: Programming for All." *Communications of the ACM* 52(11):60–67.

Roberts, C., and J. Kuchera-Morin. 2012. "Gibber: Live Coding Audio in the Browser." In *Proceedings of the International Computer Music Conference*, pp. 64–69.

Roberts, C., G. Wakefield, and M. Wright. 2013. "The Web Browser as Synthesizer and Interface." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 313–318.

Rohrhuber, J., A. de Campo, and R. Wieser. 2005. "Algorithms Today: Notes on Language Design for Just in Time Programming." In *Proceedings of the International Computer Music Conference*, pp. 455–458.

Rohrhuber, J., et al. 2007. "Purloined Letters and Distributed Persons." In *Music in the Global Village Conference* (pages unnumbered).
Smith, R. B., and D. Ungar. 1995. "Programming as an Experience: The Inspiration for Self." In *Proceedings of the European Conference on Object-Oriented Programming*, pp. 303–330.
Sorensen, A. 2005. "Impromptu : An Interactive Programming Environment for Composition and Performance." In *Proceedings of the Australasian Computer Music Conference*, pp. 149–153.
Sorensen, A., and A. Brown. 2007. "aa-cell in Practice: An Approach to Musical Live Coding." In *Proceedings of the International Computer Music Conference*, pp. 292–299.
Sorensen, A., and H. Gardner. 2010. "Programming with Time: Cyber-Physical Programming with Impromptu." *ACM SIGPLAN Notices*, 45(10):822–834.
Swift, B., et al. 2013. "Visual Code Annotations for Cyberphysical Programming." In *International Workshop on Live Programming at the International Conference on Software Engineering* (pages unnumbered).
Tanimoto, S. 1990. "VIVA: A Visual Language for Image Processing." *Journal of Visual Languages and Computing*, 1(2):127–139.
Ungar, D., and R. B. Smith. 2013. "The Thing on the Screen Is Supposed To Be the Actual Thing." In *International Workshop on Live Programming at the International Conference on Software Engineering* (pages unnumbered).
Wakefield, G., W. Smith, and C. Roberts. 2010. "LuaAV: Extensibility and Heterogeneity for Audiovisual Computing." In *Proceedings of the Linux Audio Conference* (pages unnumbered).
Ward, A., et al. 2004. "Live Algorithm Programming and a Temporary Organisation for its Promotion." In O. Goriunova and A. Shulgin, eds. *Read Me: Software Art and Cultures.* Aarhus: Aarhus University Press, pp. 243–261.
zmölnig, I m. 2007. "Patching Music Together: Collaborative Live Coding in Pure Data." In *Proceedings of the Pd Convention*. Available online at artengine.ca/~catalogue-pd/4-zmoelnig.pdf. Accessed September 2013.