The Aesthetics of Imperfection in Music and the Arts

Live Coding and Failure

20

Live coding and failure

Shelly Knotts

Emerging at the beginning of the twenty-first century, live coding is an algorithmic performance practice at the intersection of generative art and laptop performance. Performances involve the programmer writing and editing programming code to produce creative outputs including music, visuals and dance. This code takes various forms: text-based languages that describe sequences of musical notes or audio samples and associated parameters; low-level sound waves combined with filters to produce synthesized sounds; and graphical programming languages that emulate a studio patch bay, connecting various modules for sound production and the logic to change it over time.

Deleted: 21

Deleted: C

Deleted:

The process is exposed to the audience through the projection of the performer-programmer's coding interface, and performances are notoriously precarious with pressures of fast and untested coding leading to frequent onstage technical problems and coding errors. While it's not unusual for a jazz musician to suffer a broken string or reed in the course of performance, it's unlikely that their whole instrument will collapse. But for a live coder, self-built, unstable systems, programmed in real time, makes this a possibility.

Live Coding traverses notions of perfection and imperfection, using computational tools similar to those used in engineering, science and security, where system weaknesses come at great cost, and large-scale projects require months of work for large teams of debuggers to ensure system stability. Live Coding reveals the imperfection of these technical systems, putting them on live display. Even a performer with a mastery of the programming language doesn't always escape the inevitability of imperfection, when a misplaced comma or an extra '0' can be the difference between artistic success and technical failure. Navigating this technology, we become aware of the fragility of our programmed world where the stability of transport systems, communication and other human activities are supported by someone somewhere typing the underlying code without fault.

Deleted:

Deleted: ,

Failure in live coding operates on multiple levels: sociocultural, structural, musicological and performative. The inevitability and broad cultural acceptance of

Deleted: -

Deleted: ;

Deleted: ;

failure, error and imperfection in live coding leads us to re-examine musical virtuosity in light of a performance practice which relies on cognitive flexibility over the precision of physical action expected in acoustic music. These narratives of failure can also be seen as a political and social dissent. By resisting strongly defined boundaries and idealized forms of musical practice based on technical accuracy, live coding remains an inclusive, open-ended and evolving practice grounded in social and creative freedom.

<INSERT DEFINITION HERE>

As part of the registration process, participants of the 1st International Conference of Live Coding were asked to define it.[1] Diverse responses suggested that fifteen years of an artistic practice has failed to produce a consistent definition. Common descriptions in the survey included immediate, changeable, performative, emergent, generative, indeterminate, intuitive, experiential, responsive and even resisting definition. Social characteristics of the community are often described, and exposing the expressivity of programming is a recurrent concern. The use of code to formalize structural elements of the music led some to describe live coding as sitting somewhere between improvisation and composition, as performers fluctuate between coding moment-to-moment aspects of sound and longer musical arcs. The relationship between performer and code is sometimes described as that of a master programmer exerting control over the developing programme. Others think of the performer more passively, responding to outputs of an emergent system. In reality diverse practices encompass a wide spectrum of interaction between these two extremes. These descriptions partly reflect the content and ethos of the TOPLAP draft manifesto,[2] which was written in 2004 by the first wave of live coders in an attempt to loosely define the emerging practice. The manifesto still exists as an open-ended document, and can theoretically be adapted or rewritten.

The live coding community's resistance to a singular definition is congruent with narratives of non-idiomatic improvisation, for which '[d]iversity is its most common characteristic'.[3] Artforms with porous boundaries defy quantification, challenge productivity and circumvent ideologies of success and genius. Open-endedness is a recurrent theme in performance art since the 1960s, and explorations of failure in this genre often focus on exposing alternatives to treading known paths to achieving cultural perceptions of success.[4]

From live debugging of a piece of running code to awareness among artists and audiences that live coding is an error-prone practice, failure is ever-present. With its basis in tools of capitalist production, we can consider live coding under the lens of wider cultural theories of failure. Sandage traces the linguistic and cultural routes of

Deleted: s

Deleted: ,

Deleted: →

Commented [A1]: AQ: Please provide either the heading in place of 'INSERT DEFINITION HERE' or fill in the missing text, if any.

Commented [MOU2R1]: This section title is correct. It is a reference to coding and I believe is clarified by the first sentence of the section.

Deleted: 0.

Deleted: :

Deleted: ,

Deleted: s

Deleted: ,

failure, moving from the economic domain into the social, but always considered as the antithesis to success. Success, in its capitalist formation is rigid in its definition, determinates from the outset and excludes that which is unknown and explorative.[5] Halberstam reclaims failure as an opportunity to reject normative social roles.[6] Programming as a productive activity that drives modern economies is reformed in live coding as a transitory act with no product, aim or end point. Traditionalist pursuit of perfection in the classical arts leaves only one path to success; in improvisation there are many ways to fail, and live coding makes these uniquely visible.

This chapter considers narratives of failure in live coding practice and culture. I analyse technical tools which foreground indeterminism and imprecision in live coding practice and explore how error may structure performances. I consider the position of the performer in a failure-prone artistic practice, and the social function of error in performances.

Error as musical structure

As a practitioner I find my live coding performances are punctuated more obviously by my failures – technical errors or algorithms with unexpected outputs – than by accuracy in the rendering of musical ideas into code. Debugging incorrectly typed code often means a section extending longer than would be musically coherent, while algorithms with unexpected outputs may require changing trajectory to ensure musical flow. The most dramatic live coding motives include software crashes which cause an unmusical mid-set pause or abrupt endings.

The decoupling of embodiment from immediate musical action that occurs when using code to describe musical events and structures means that conceptual errors may also occur, where live coder makes errors in the complex task of turning a musical idea into code.[7] These kinds of failures introduce a level of indeterminacy, necessitating the following of unforeseen musical paths, constantly rethinking the next move. This process can be differentiated from well-understood aesthetic and musical errors in improvisation, such as playing the wrong note in a jazz performance, which can be quickly folded into the musical structure through repetition, re-emphasis or a change of direction. Distinct timelines of physical and musical action in live coding means living with one's failures, sometimes for longer than one would like.

The next section dissects how failure and error function structurally in live coding performance practice and how this differs from error handling in other improvisation contexts. I explore how a live coding performer's inevitable inability to algorithmically describe a musical idea is a unique aspect of algorithmic music practices

Deleted: ,

Deleted: ,

Deleted: ,

Commented [A3]: AQ: Per template style, the head numbers are to be deleted. Please confirm if this style followed for consistency across the book is fine.

Commented [LP4]: Yes, this is fine for consistency across the book.

Deleted: 1.

Deleted: -

Deleted:

Deleted: ,

and live coding in particular which highlights the imperfect ability of the human performer to render musical imagination in code lines.

Conversing with failure

Formatted: Font: Not Italic

Formatted: Font: (Default) Times New Roman

> The distance between what one may expect an algorithm to have as effect and what the algorithm actually brings about is irreducible; this gap is the thread along which conversational sound programming is kept moving.[8]

Rohrhuber and De Campo's model of live coding describes performances as unfolding along a trajectory of repetitive failure. A performance begins with an imagined musical goal state which the performer then attempts to realize into computing code. This begins a feedback loop which continues for the duration of the performance between intended and actual (often unpredictable) musical outcomes of coding.

Deleted: s

Rohrhuber and De Campo describe this temporal interplay as a live dialogue with failure, requiring a continuous rewriting of an imagined performance trajectory.[9] The importance of feedback in musical improvisation – error often requires a change in musical process and context – is noted by Pressing, who defines two possible strategies for continuing an improvisation: 'associative' and 'interrupt' generations. In 'associative generation' the majority of the strong sound elements of the improvisation continue and variation takes place by making a weak sound element stronger or introducing an additional strong element. In 'interrupt generation' a significant number of strong elements are replaced with new musical characteristics signalling a new section in the performance.[10]

In live coding these generative strategies can be seen in response to failing to accurately translate a musical idea into code. The performer reacts to the output of code they write by either refining or redefining the imagined musical goal state, and ultimately the trajectory of the performance. In the case of refining, the performer reacts to the outcome (O) of failing to formalize their intentions by revising the code – making small edits – in an attempt to get closer to the original goal. It may take several revisions ($R_1 \, . \, . \, . \, R_n$) to get satisfactorily close to the original intended goal state (G), with the result that the imagined performance trajectory is temporally stretched by repeated failings. In the case of redefining, the performer reacts to unintended outcomes (O) of the code they have written by accepting it as the starting point of a new, though unintended, trajectory, and imagining a new musical goal state (G), taking the performance in a previously unplanned direction. In both cases, failure performs a structural function, either to elongate or to divert the planned trajectory of the performance.

Deleted: s

Deleted: ..

In Jonas Hummel's text score for live coders *Mimickry*, performers are directed to mimic and outdoor soundscape. After a short period of listening they code the sound they hear. The piece externalizes the goal state of the improvisation, making audible the difference between intention and result. The audience perceives the process of refining a piece of code towards a goal sound state.

Deleted: s

Magnusson's *Encoding the Marimbist* also harnesses the mistranslation of intentions into computing code as a performance process by using a human performer to carry out code-like instructions. Magnusson writes instructions (which include musical and physical directions), which marimbist Greta Eacott attempts to follow accurately as possible. Magnusson edits existing 'code' or writes new directions in response to her (mis)interpretations. The piece unfolds with both performers navigating interpretation inaccuracies and the indeterminism of the output of a human performer following code-like instructions.

Transient versus persistent errors

Formatted: Font: Not Italic

Deleted: .

Formatted: Font: (Default) Times New Roman

In non-idiomatic improvisers, error and imperfection are familiar topics, as Bailey describes: '[the] accidental can be exploited through the amount of control exercised over the instrument, from complete – producing exactly what the player dictates – to none at all – letting the instrument have its say.'[11] This continuum between determinacy and indeterminacy, control and 'letting go', is often present in live coding where the complexity of an algorithm and its level of interaction with other running processes can vary the predictability of results.

Deleted: .

Though both practices embrace imperfection, I argue that error impacts musical structure differently in non-algorithmic performance practices. Transient errors, commonly seen in instrumental improvisation, require performers to return to the error, and variations of it, incorporating it into the structure of the performance. As soon as the performer stops their actions causing the error, the state of the performance returns to as it was before. In live coding, a performer who encounters error, such as a difficult-to-solve syntax problem, or a full-on system crash, lives with that error as part of their performance until they are able to locate and fix the error (or else divert the performance in the direction of the error). Non-action on the part of the performer results in the error persisting. A difficult-to-locate syntax error lasts for the duration of the performer's debugging time, and a system crash lasts for however long it takes to restart the software or laptop.

The next section considers the performer in live coding and how the practice may lead us to reconsider virtuosity and musicianship.

The virtuosic live coder

Cox and McLean describe the term 'virtuoso' as emerging from Aristotle's ethics.[12] To be virtuous is to act well in the world, which can only be achieved by practice. In instrumental music, virtuosity is commonly related to physicality, accuracy and precision: the musician develops their muscle memory such that more and more complex figures are intuitive. The skill of the performer relates to rendering scores or making improvised responses with accuracy and control.

Ostertag examines virtuosity in computer music, citing theremin playing and turntablism as the only acts of electronic music-making which come close to virtuosity, as the time taken to learn the skill and the granularity of physical control required display virtuosic traits.[13] In contrast, much live coding literature describes the live coder as virtuosic, with skills comparable to those of an instrumental performer.[14] However, lack of relation between physical action, muscle memory and resulting sound in live coding leads us to consider the term 'virtuoso' as describing the cognitive reasoning abilities of the performer, rather than physical precision.

> live coders may be less susceptible to the influence of habit and mechanical modes of musical expression because the motor skills required are less complex than instrumentalists and consequently less of their behaviour is activated outside of their perceived conscious control or awareness. . . . The instrumentalist has . . . a vast resource of pre-encoded units of behavior which will sustain their music making once they have relinquished conscious control, which . . . is not available to the live coder . . . [who] would probably struggle to talk and code at the same time. It therefore seems plausible that live coders require at their disposal, more cognitive resource to engage conceptual processes and process memory, which will tend to produce behaviour that is reactive rather than reflexive.[15]

Undoubtedly practice and in-depth knowledge of the system increases the abilities of the live coder to type faster and more accurately programme their musical intentions. However, this is a process with a high degree of imprecision and a great deal more cognitive awareness than the intuitive behaviours of instrumental performers.

Rohrhuber and De Campo propose that the separation of physical action and sound production in algorithmic performance points towards a different conception of virtuosity: 'Because the necessary combination of timeless algorithm and algorithmic process impedes phantasms of immediate total control, the common notion of the virtuoso as the prototypical artist becomes but one possibility in a broader field. Neither the machine needs to be virtuosic, nor the human performer.'[16]

Deleted: 2.

Deleted: ,

Deleted:

Deleted: …

Deleted: …

Deleted: …

Deleted: …

Deleted: …

Deleted: ,

Cocker applies the term *mêtis* to the skills of the live coder which relates to a reactive type of intellect, involving acting within emergent or unpredictable systems: '*mêtis* described a form of wily intelligence capable of seizing the opportunities made momentarily visible as the prevailing logic within a given structure or system yields. *Mêtis* is the art of preparing for what could not have been anticipated or planned for in advance.'[17]

Deleted: .

Live coders comment that 'sometimes it is hard enough just to get a program running at all let alone somehow to have anticipated enough about the likely program flow to make significant moves'.[18] If we acknowledge the inherent indeterminacy of working with algorithmic systems, 'code both performs and is performed through the practice of coding in real-time. Both mutually create and define each other, inter-acting in indeterminate and uncertain ways.'[19] then Cocker's description of the live coder as a highly skilled reactant seems more accurate than Aristotle's virtuoso.

Deleted: ,

Deleted: -

Parkinson and Bell offer an alternative view on virtuosity in live coding: 'virtuosity on the part of a live coder might instead manifest itself through the use of difficult or less-common functions, complexity of code written live, or elegance of expression in coding'.[20] However, the temporal nature of live coding means the performer must balance the time cost versus complexity benefit of any code writing. In time-limited performances, writing complex algorithms may not be the most efficient strategy.

My experience of live coding reflects descriptions, where working with and adjusting quickly to the unexpected is a greater aspect of a performance than writing complex algorithms.[21] Stringing together multiple simple lines of code which interact in complex and unpredictable ways is one strategy to balance the time versus complexity trade off. This technique is highlighted in the 9-minute blank slate live coding form of the Mexican live coding community, in which extreme time limits favour simplicity of algorithmic design over complex structures.

Deleted: .

Deleted: nine

Precision and accuracy are often not prerequisites of a live coding performance. The algorithmic nature of live coding allows the possibility to design performance systems which give partial control of the performance to the software. Kiefer's *Approximate Programming* aptly demonstrates this:

> The system is named approximate programming because the results for a particular gesture or gene are, at least initially, unknown. . . . The system is primarily exploratory, although exploration transitions towards repeatable performance as the search space is mapped out by the player.[22]

The *Approximate Programming* software generates more complex soundscapes by automatically combining small chunks of code. The low-level parameters of the resulting sound can then be explored using a custom physical interface, or edited directly by the performer using the coding interface. The system can produce musical complexity with relative speed, but in the process sacrifices a high level of performer autonomy and the predictability that comes with instrumental playing. Although the sound output is partially predictable, as the code used will produce a particular range of sounds, the system is primarily exploratory with the performer responding to new combinations of sounds and parameters.

Playing with unpredictability, the 'autocode' function in Magnusson's *Ixi Lang* function produces a random set of code lines which are automatically evaluated. Each line consists of an instrument with a pattern of notes. The function can be used to create a new section of the performance, where the performer creates sense of the newly created 'random' code and sound.

Hummel's *Mind Your Own Business* uses collaboration to mediate control. *Republic* – a networked collaborative coding tool in the programme SuperCollider – is used to divide the coding between players so that each is responsible only for one element: rhythm, pitch, mixing and so on. All programming is collaborative and no player has full knowledge of the code that other performers are writing. Code edits can produce sounds that cannot be predicted from the code on their own computer.

The virtuosity of a master musician often describes their ability to master the physical control of their instrument, with precision and accuracy. Live coding's unpredictability and imprecision requires a different skill set including decisiveness, creative problem-solving and the ability to create structure with unpredictable systems. Following Halberstam (see the first section), we could position the live coder, not as virtuoso but as developing a mode of musicianship outside the scope of normative narratives of skilled music-making. The next section explores how the culture and practices of live coding, embrace error and failure as part of the performance narrative.

'Show us your errors!'

Much has been written about the practice of showing one's screen in live coding.[23] It is customary to project your coding interface on a publicly visible surface in the performance space. 'Show us your screens' was a proposition in the TOPLAP manifesto, and has been a much debated topic in the live coding community. Some argue that it's important to share the process of the performance, while a counterargument contends that the lack of code literacy in audiences makes screen projection meaningless.

Deleted: or

Deleted: , etc.

Deleted:

Deleted: 0

Deleted: ,

Deleted:

Deleted: 3.

Deleted: U

Deleted: Y

Deleted: E

Deleted:

Rohrhuber and De Campo read showing your screen as an ‘ally of irritation’, helping to build a community among performers and audience members around code debugging.[24] My focus here is on the importance of the practice in revealing the narrative of error to audience members.

There are several ways that error is perceived through screen projection: errors messages, pauses in on-screen activity and visibly obvious system failures. The projected screen shows the performer interface, which normally includes some system of highlighting errors in the code. Usually this is made visually obvious to aid the performer in correcting their code. For example, in SuperCollider, information is printed in a message window preceded by capitalized ‘ERROR’ text. In Sonic Pi error messages appear highlighted blue or pink, depending on the type of error.[25] A pause in on-screen activity may point towards a technical issue for the performer as they think through an arising problem or deal with off-screen hardware problems. Serious errors, which cause system crashes, resulting in the need to restart software, are often visibly obvious. Through screen projection the performer’s debugging becomes part of the narrative, as visual cues give the audience insight into disruptions to the flow of the performance. Di Prospero discusses collectivity in the live coding scene and its desire for openness, whether it be sharing the code by projecting, openness to new forms of live coding or the easy acceptance of beginner live coders.[26] This is in part driven by the ‘show us your screens’ ideology as this exposes the fallibility of even expert live coders and facilitates collective error fixing. The acceptance of error and the open culture of facilitates communication between audience members and between performer and audience. It is acceptable for audience members to point out errors to a struggling performer, and audience members may discuss the potential causes of an error or failure.

Crashing is a further symptom of cultural acceptance of failure. In most computer music scenes, stability is seen as important, and crashing should be avoided. In live coding, performers may take precautions to avoid crashing – monitoring the processing load of the software and computer and so on – but the instability of the software and running of untested code mean that crashing is far more likely than in other computer music systems. One trope of live coding performances is to end by overloading the system causing it to crash. Languages such as Ixi Lang have inbuilt crashing functions that performers can use to cause a software crash at an indeterminate future time;[27] a Max/MSP based live coding community in The Netherlands built a practice around crashing at the end of the performance.[28]

The acceptance of crashing in live coding was clear in one of my own

Deleted: s

Deleted: ,

Deleted: , etc.

Deleted:

performances, which crashed catastrophically mid-set. The cause was a hardware failure, but I was so used to dealing with coding and software errors in my sets that I assumed, as did the audience, that the problem was code-related. At the point the crash happened I had been building up slowly to the point of dense rhythmic structures for 30 minutes. The break in my set lasted 15 minutes, while I located the source of the problem and restarted my system. After that I started coding again from scratch as I neither had a saved version of the code nor saw any musical sense in taking up where I left off. Though I was frustrated that the performance had not unfolded as intended, the break gave me an obvious marker to change course and try something different in a way that might not have made sense otherwise. After my set I had conversations throughout the evening with other performers and audience members, who were sympathetic to how my set unfolded, and assumed that the problem was code-related. Later in the evening – after another performer had also had a mid-set crash – the gig organizer remarked that crashing is 'just what happens' in live coding.

Rewriting the rules and failing to follow them

Although a term with problematic beginnings, Brock redefined 'Degenerate Art' positively as exploring the social rules of what is expected from art and breaking them to present something which falls outside the rule-set as a work of art for aesthetic appreciation – usually critiquing social values in the process.[29] Programming is usually functional, technical and productive; it contributes to the economy and a large part of the post-industrial labour market. However, live coding presents error-driven, non-productive coding activity as an experiential leisure activity, free from the necessity for technical innovation or stability. The programming is ephemeral and lacks output which would give it economic value. Likewise, live coders fail to live up to normative narratives of the successful artist, creating a new path of creativity through interacting with unpredictable and indeterminate systems.

Halberstam's theory of failure describes how its antithesis, 'success', came to describe treading a known path towards a pre-defined mould, where normative societal values defined personal success or failure. For Halberstam, failure is rejecting the normative in favour of exploring the other or the unknown, bringing with it new freedoms to define one's own rule-set for life, values, art or economics.[30]

The live coding community rejects rigid definitions of its own practice, favouring instead inclusivity, multiple pathways, and evolving practices and descriptions. Live coding performances are structured by the performer's engagement with their own failure. Screen projection displays this narrative to audiences, building a

Deleted: thirty

Deleted: fifteen

Deleted: s

Deleted: 4.

Deleted: ,

Deleted: ,

Deleted: ,

community of problem solvers and an acceptance of failure and error as inevitable for even the most experienced performers. Live coding fails to follow the expectations of programming, and live coders do not conform to the normal formula for performative success. Live coding rewrites the rules of what is accepted as performance, community, practice and artform, and then rewrites them again when unforeseen pathways emerge.

References

2. Aaron, S., Blackwell, A., Hoadley, R. and Regan, T. (2011), ‘A Principled Approach to Developing New Languages for Live Coding’, in A. Jensenius, A. Tveit, R. Godøy and D. Overholt, eds, *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 381–6.

3. Bailey, D. (1993), *Improvisation: Its Nature and Practice in Music*, New York: Da Capo Press.

4. Blackwell, A. and Collins, N. (2005), ‘The Programming Language as a Musical Instrument’, in *Proceedings of PPIG05* (Psychology of Programming Interest Group).

5. Brock, B. (2010), ‘Cheerful and Heroic Failure’, in L. Le Feuvre, eds, *Failure*, Cambridge, MA: MIT Press/Whitechapel Gallery, pp. 180–2.

6. Brown, A. and Sorensen, A. (2009), ‘Interacting with Generative Music Through Live Coding’, *Contemporary Music Review*, 28: 1, pp. 17–29.

7. Cocker, E. (2015), ‘Live Coding / Weaving — Penelopean Mêtis and the Weaver-Coder’s Kairos’, in A. McLean, T. Magnusson, K. Ng, S. Knotts, and J. Armitage, eds, *Proceedings of the First International Conference on Live Coding*, ICSRiM, University of Leeds.

8. Collins, N. (2011), ‘Live Coding of Consequence’, *Leonardo*, 44: 3, pp. 207–11.

9. Cox, G. and McLean, A. (2013), *Speaking Code: Coding as Aesthetic and Political Expression*, Cambridge, MA: MIT Press.

10. Cox, G. (2015), ‘What Does Live Coding Know?’, in A. McLean, T. Magnusson, K. Ng, S. Knotts, and J. Armitage, eds, *Proceedings of the First International Conference on Live Coding*, ICSRiM, University of Leeds.

11. Di Prospero, C. (2015), ‘Social Imagination’, in A. McLean, T. Magnusson, K. Ng, S. Knotts, and J. Armitage, eds, *Proceedings of the First International Conference on Live Coding*, ICSRiM, University of Leeds.

12. Halberstam, J. (2011), *The Queer Art of Failure*, Durham, NC: Duke University Press.

13. Kiefer, C. (2015), ‘Approximate Programming: Coding Rough Gesture and Numerical

Deleted: ,

Deleted: ,

Commented [A5]: AQ: Please provide publisher details for this reference.

Deleted: .

Deleted: -

Deleted: 18

Deleted: .

Deleted: -

Commented [A6]: AQ: Please provide publisher details for this reference.

Deleted: -

Deleted: 2

Deleted: ,

Commented [A7]: AQ: Please provide publisher details for this reference.

Deleted: *Proceedings of the First International Conference on Live Coding*.

Commented [A9]: AQ: Please provide publisher details for this reference.

Deleted: in *Proceedings of the First International Conference on Live Coding*

Processes', in A. McLean, T. Magnusson, K. Ng, S. Knotts, and J. Armitage, eds, *Proceedings of the First International Conference on Live Coding*, ICSRiM, University of Leeds.

14. Le Feuvre, L. (2010), *Failure*, Cambridge, MA: MIT Press/Whitechapel Gallery.

15. Lee, S. W. and Essl, G. (2015), 'Web-Based Temporal Typography for Musical Expression and Performance', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 65–9.

16. Magnusson, T. (2014), 'Herding Cats: Observing Live Coding in the Wild', *Computer Music Journal*, 38: 1, pp. 8–16.

17. Ogborn, D. (2014), 'Live Coding in a Scalable, Participatory Laptop Orchestra', *Computer Music Journal*, 38: 1, pp. 17–30.

18. Ostertag, B. (2002), 'Human Bodies, Computer Music', *Leonardo Music Journal*, 12, pp. 11–14.

19. Parkinson, A. and Bell, R. (2015), 'Deadmau5, Derek Bailey, and the Laptop Instrument – Improvisation, Composition, and Liveness in Live Coding', in A. McLean, T. Magnusson, K. Ng, S. Knotts, and J. Armitage, eds, *Proceedings of the First International Conference on Live Coding*, ICSRiM, University of Leeds, pp. 170–8.

20. Pressing, J. (1988), 'Improvisation: Methods and Models', in J. Sloboda, ed., *Generative Processes in Music: The Psychology of Performance, Improvisation, and Composition*, Oxford: Clarendon Press/Oxford University Press, pp. 129–78.

21. Purcell, A., Gardner, H. and Swift, B. (2014), 'Visualising a Live Coding Arts Process', in *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: The Future of Design*, pp. 141–4.

22. Rohrhuber, J. and de Campo, A. (2009), 'Improvising Formalisation: Conversational Programming and Live Coding', in G. Assayag and A. Gerzso, eds, *New Computational Paradigms for Computer Music*, Delatour, France: Ircam-Centre Pompidou, pp. 113–24.

23. Sá, A. (2018), 'Designing Musical Expression', in L. Ribas, A. Rangel, M. Verdicchio and M. Carvalhais, eds, *xCoAx 2017 — LISBON: Proceedings of the Fifth Conference on Computation, Communication, Aesthetics & X*, pp. 226–40.

24. Sandage, S. (2005), *Born Losers*, Cambridge, MA: Harvard University Press.

25. Sayer, T. (2015), 'Cognition and Improvisation: Some Implications for Live Coding', in A. McLean, T. Magnusson, K. Ng, S. Knotts, and J. Armitage, eds, *Proceedings of the First International Conference on Live Coding*, ICSRiM, University of Leeds.

26. Sorensen, A. (2009), 'Impromptu: An Interactive Programming Environment for Composition and Performance', in *Proceedings of the Australasian Computer Music Conference*, Queensland University of Technology, Brisbane, Australia.

27. Toplap (n.d.), 'ManifestoDraft', *Toplap*, website

Deleted: ,

Commented [A11]: AQ: Please provide publisher details for this reference.

Deleted: in *Proceedings of the First International Conference on Live Coding*

Deleted: -

Deleted: 6

Deleted: ,

Deleted: -

Deleted: -

Deleted: -

Commented [A13]: AQ: Please provide publisher details for this reference.

Commented [A14]: AQ: Please provide publisher details for this reference.

Deleted: in *First International Conference on Live Coding Proceedings*,

Deleted: -

Deleted: 17

Deleted: -

Deleted: 1

Formatted: Font: (Default) Times New Roman, Italic

Deleted: -

Deleted: 14

Deleted: .

Deleted: :

Deleted: -

Deleted: 1

Deleted: .

Deleted: -

Deleted: 2

Commented [A15]: AQ: Please provide publisher details for this reference.

Deleted: in *Proceedings of the First International Conference on Live Coding*

Commented [A17]: AQ: Please provide publisher details for this reference.

Formatted: Font: Not Italic

Formatted: Font: Not Italic

Formatted: Font: Italic

(http://toplap.org/wiki/index.php?title=ManifestoDraft&amp;oldid=3928), [Accessed 2 August 2019].

Deleted: /8/

Figure 20.1 Carol Breen, Antonio Roberts and Maria Witek performing at a live coding event in Coventry, UK, 2019. The code of one of the performers is projected on a screen behind them.

Figure 20.2 Refining and redefining methods of continuing a live coding performance.

Deleted: R

Figure 20.3 A live coder's screen showing an error.

[1] The full list of responses can be viewed here: https://docs.google.com/spreadsheets/d/1ogux4mAIUbXOjiFqw7fhh6WL0FKHdVvvuijEeZl5jCM/edit?usp=sharing.
[2] Toplap (n.d.).
[3] Bailey (1993), p. 83.
[4] Le Feuvre (2010).
[5] Sandage (2005).
[6] Halberstam (2011).
[7] Sá (2018).
[8] Rohrhuber and De Campo (2009), p. 123.
[9] Rohrhuber and De Campo (2009).
[10] Pressing (1988).
[11] Bailey (1993), p. 100.
[12] Cox and McLean (2013).
[13] Ostertag (2002).
[14] Aaron et al. (2011), Blackwell and Collins (2005), Brown and Sorensen (2009), Magnusson (2014), Ogborn (2014), Parkinson and Bell (2015), Sorensen (2009).
[15] Sayer (2015), n. p.
[16] Rohrhuber and De Campo (2009), p. 116.
[17] Cocker (2015), n. p.
[18] Collins (2011).
[19] Cox (2015), n. p.
[20] Parkinson and Bell (2015), n. p.
[21] Collins (2011), Cox (2015), Rohrhuber and De Campo (2009).
[22] Kiefer (2015), n. p.
[23] Magnusson (2014), Purcell, Gardner and Swift (2014), Lee and Essl (2015).
[24] Rohrhuber and De Campo (2009).
[25] See https://sonic-pi.net/
[26] Di Prospero (2015).
[27] See http://www.ixi-audio.net/ixilang/
[28] See https://cycling74.com/
[29] Brock (2010).
[30] Halberstam (2011).