Live Coding

A User's Manual

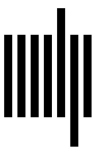
By: Alan F. Blackwell, Emma Cocker, Geoff Cox, Alex McLean, Thor Magnusson

Citation:

Live Coding: A User's Manual
By: Alan F. Blackwell, Emma Cocker, Geoff Cox, Alex McLean, Thor Magnusson
DOI: 10.7551/mitpress/13770.001.0001
ISBN (electronic): 9780262372633
Publisher: The MIT Press
Published: 2022

OA Funding Provided By:

OA Funding from Author



The MIT Press

2 Partial Histories

Instead of attempting to establish a historical canon for live coding, this chapter recognizes that live coding is a way to question assumptions of historicism, as well as liveness and temporality, which are addressed in later chapters.[1] We approach the idea of *partiality* through the concept of Donna Haraway's *situated knowledges*: we do not adopt a universal perspective but recognize that knowledge is produced in lived realities and in doing so makes our partiality transparent.[2] So in calling this chapter a partial history (or rather, *histories* in the plural), we draw attention to how histories are created. As such, we urge readers to remain skeptical of how we have pieced together some of the details of the development of live coding and how we have made sense of events *after the fact*. Going beyond the contingency of any history (given the fallibility of human memory, the instability and partiality of events, and the complexity of emerging digital narrative forms and the forces that shape them), a straightforwardly historicist approach must be questioned in relation to the fundamental liveness of live coding.[3]

The narrative history of live coding that we have assembled in this chapter has therefore been prepared in a manner that echoes live coding itself, using a process that has been collaborative, experimental, improvised, and reflective.[4] Moreover, the text that you are reading now draws on the subjective experiences of key practitioners and their memories—embracing the stories *and* the falsehoods—and was developed through a series of interviews back in 2016 with those whom we considered to be key figures in the live coding community at the time. They included some of the authors of this book, who were interviewed by other authors.[5] Based on these interviews, the individuals who appear in the following sections (with initials that will be used when referring to our thematic analysis) are AA: Amy Alexander; AC: Alexandra Cardenas; AdC: Alberto de Campo; AM: Alex McLean; AS: Andrew Sorensen; BatMan: Benoit and the Mandelbrots; DG: Dave Griffiths; DO: David Ogborn; GW: Ge Wang; JA: Joanne Armitage; JR: Julian Rohrhuber; KS: Kate Sicchio; MH: Mike Hodnick; NC: Nick Collins;[6] SA: Sam Aaron; SK: Shelly Knotts; TM: Thor Magnusson. Our choice of interviewees takes

in perspectives of the genesis of the live coding movement in Europe and the US and subsequent import and development in Mexico and elsewhere. We are alert to questions of diversity in choosing such a sample. Women were engaged in live coding from its inception, although in smaller numbers than men. We note in particular that this list reflects the limited ethnic diversity (at least in Europe) of the early community.[7] As live coding techniques have recently found wider adoption, this picture has somewhat changed, as reflected in the variety of expositions shared in the following chapter.

Live Coding Narratives and Nonlinear Histories

Before reflecting on the specific histories of live coding, it is worth briefly considering a wider contextual frame for creative coding, specifically in relation to computer music, from which live coding initially emerged. The history of computer music extends back to 1956 when Lejaren Hiller and Leonard Isaacson began working on their *Illiac Suite*, applying Markov models to generate a musical score. In the following decades, composers and computer scientists wrote bespoke software for score generation and synthesis, but none produced sound in real time, due to the technical limitations of that generation of computers. A key trajectory is mapped through the series of languages, known as MUSIC-N, created by Max Mathews, which along with its descendants, such as CSound, provided the theoretical foundations for much of the software described in this book. By the late 1980s when the period covered by our interviews starts, *computer* music was primarily based on sequencing the notes played by hardware synthesizers and samplers rather than real-time software synthesis. Commercial software design was aimed at simulating recording studios and was rarely used in real-time performance. Software enabling musicians to write algorithmic music, such as Max and Open Music,[8] developed at IRCAM (Institute for Research and Coordination in Acoustics/Music) in Paris, started to emerge from research and academic environments, but it was in the 1990s that mass-produced computers began to be fast enough for real-time audio processing. Programming languages with a focus on real-time audio processing, such as Miller Puckette's Pure Data and James McCartney's SuperCollider, were able to directly synthesize sound from algorithms and filter and combine incoming audio streams and/or stored samples.[9] Later in this chapter, we focus on the pivotal moment at which these technologies intersected with interactive approaches to programming. While real-time audio processing still required relatively expensive computers, by the mid-1990s, individuals were beginning to be able to do this type of work on high-end laptops. By the late 1990s, a new electronic music scene had emerged where people would bring laptops into clubs and pubs, placing themselves (or being placed, as performers) on the stage with their laptop.[10]

In most of these early performances, there was little to see: audiences simply watched a performer sitting on stage without visuals or any gestural movement. The initial idea of the practitioners was that—as electroacoustic musicians had claimed three decades earlier—people would get used to the *nonevent*, or rather the nonvisual event, of the performance. However, the tendency of the audience to keep watching for something, even if just the movement of a mouse, made it obvious that there was space for other things to happen on stage. As video projectors became cheaper, new responses to this opportunity came within reach, including VJing and abstract visualization of the audio signal.[11] Most performance venues became equipped with projectors set up to project onto the stage. Although there are precedent cases of live coding in the arts—such as the Hub in music, Larry Cuba in the visual arts, and diverse experiments with live programming or interactive programming in computer science—it is really the foundation of early 2000s electronic music that becomes the context of live coding's growth as an artistic form, which now includes music, visual arts, dance, and other creative forms.

The 1980s: A Generation of Geeky Artists

The gradual expansion of computer-based electronic music (since the 1980s) and increased cultural interest in creative coding (since the turn of the millennium) both helped to shape the wider conditions from which live coding evolved. However, the personal histories of prototypical live coders reveal a distinctive pattern, which often combined early exposure to technological development and access to creative contexts within which to experiment. The first generation of live coders were children of the late 1970s and 1980s, for whom the computer was both a promise and an obligation. Where they could afford them, their parents bought home computers, but the potential of these machines was often just out of reach. Time spent with ZX Spectrums, TRS-80s, CP/M microcomputers, and Commodores was often grabbed from an older sibling or shared with an eager parent. There were few computer games, so you had to write your own. But with no internet to turn to for advice, the text laboriously copied out of magazines seldom worked. The opportunities for creative expression were often little more than simple beeps or colored blobs on the screen.[12] The "random" [RANDOMIZE, or RND] key word was an essential escape from complete boredom.

Schools in the 1980s were also investing in Apple IIs and BBC Micros, but creative coding was not the first priority in computer science education. Given the relatively limited resources, education in computing was task focused rather than imaginative or playful.[13] Whether or not they had creative experiences with computers, many live coders did have creative childhoods. Many headed for careers in the professional arts through early promise as instrumental musicians, dancers (KS), or composers (SK, MH).

Some won prizes, formed bands, and performed as soloists on piano (NC, AA), trumpet (AS), or percussion (AC) or switched to guitar (TM, AdC) or saxophone (SA) so they could form rock and jazz bands. AM used school equipment to experiment with a drum machine. At the same time, some also followed more stereotypically technical interests. SA became obsessed with programming his graphing calculator, JR coded on paper because he didn't have a computer, and DO, whose home computer had limited software, wrote C code in notebooks with a pencil until his parents finally bought a C compiler for his birthday.

This ensemble of *geeks* and musician-technologists did not meet each other,[14] or learn there were others like them, since children had no access to the internet in the 1980s. DO remembers a 1985 computing camp where the most memorable experience was not writing code but sending his first email. This "weird" combination of interests—both art and technology—was sufficiently unusual that without access to others via the internet, live coders often grew up working in isolation from a wider sense of a community of practice. It was years before AS (based in Australia) could experience related work. Even after hearing of the London live coding scene and recognizing that it was the same thing he was attempting, he could find only a single online video of live coding (Ge Wang and Perry Cook using ChucK).

The Art School Shapes Digital Culture

For some future live coders, it was a challenge to find contexts for combining the technical and the creative.[15] However, their musical and performance talent led many of this first generation of live coders to art school, and they all remember the tutors who first inspired them to experiment with generative systems and digital media. Notable figures who influenced and inspired our nascent live coders included Larry Cuba and Morton Subotnik at the California Institute of Arts (AA), William Latham's recruitment of students from Bournemouth University (DG), Martin Robinson at Middlesex University (NC, TM), Perry Cook and Paul Lansky at Princeton University (GW), Juan Reyes at the University of Colombia (AC), Kurd Alsleben at the Hochschule für bildende Künste in Hamburg (JR), Andrew Brown at Queensland University of Technology (AS), Helmut Dencker at the Hochschule für Musik in Graz (AdC), and Scott Wilson at the University of Birmingham (SK). It was clear to both tutors and students that conventional scripted notation, in the digital era, might be a mechanical constraint foreign to the creative opportunities of free improvisation and the process-based and experimental arts. As the first digital generation of students were exposed to these practices, they and their tutors seem to have realized together that code could be treated as either a medium, a craft material, a composition system, or a performance notation—principles that were

increasingly explored in degree show pieces or in new societies, collectives, and collaborations between tutors and students. Most of this activity took place in graduate arts programs around the mid-1990s, where access to sufficiently advanced hardware and software, and much time for exploration at hand, enabled tutors to teach experimental classes in electroacoustics (for musicians), computer music (for sound engineers), or interactive and media art (for graphic designers).

Slub

One of the most influential of these collaborations was between two people who initially met as students: Adrian Ward, when completing a hybrid arts-computing degree (bachelor of science in media lab arts), and Alex McLean, who was frustrated by the lack of creative opportunity in his applied computing course (bachelor of science in computing), both at the University of Plymouth in the UK (where Geoff Cox was also teaching). Ward had concentrated on the subversion of standard digital media tools in his program *AutoShop* (1999), which offered a satirical commentary on the functionality of Adobe's Photoshop.[16] Ward and McLean's shared desire for programming to be a creative activity was revived when they met again after both had graduated and moved to London.

They became regular collaborators, exploring the relationship between packaged and improvised software in "The Generative Manifesto," a precursor to the TOPLAP manifesto that was offered to the public in a Perl community event at the Institute of Contemporary Arts (ICA) in London in September 2000, during which Ward shouted the manifesto over a soundtrack of rhythms synthesized by McLean. The two soon gained an international profile, with Ward winning the software art prize at Transmediale in 2001 for his next project AutoIllustrator (2001) and McLean winning the same prize the following year for forkbomb.pl (2002)—a simple script that would visualize the process of crashing the machine it was run on.[17]

The Generative Manifesto

Performing under the name slub,[18] Ward and McLean's "Generative Manifesto," as preserved in the handwritten original reproduced here, advocated:

1. Attention to detail—that only hand-made generative music can allow (code allows you to go deeper into creative structures);
2. Realtime output and compositional control—we hate to wait (it is inconceivable to expect non-realtime systems to exhibit signs of life);
3. Construct and explore new sonic environments with echoes from our own. (art reflects human narrative, code reflects human activity);

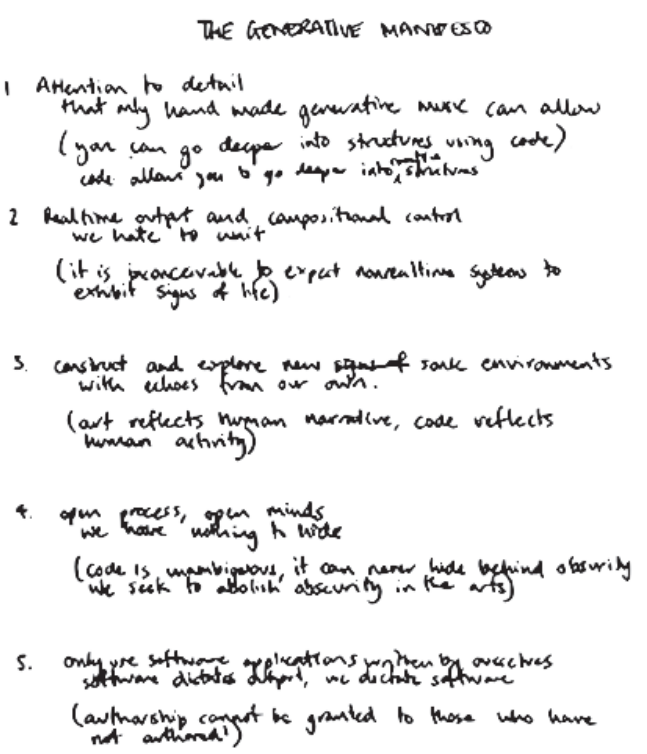


Figure 2.1
The original handwritten slub "Generative Manifesto."
Source: From Adrian Ward and Alex McLean.

4. Open process, open minds—we have nothing to hide (code is unambiguous, it can never hide behind obscurity. We seek to abolish obscurity in the arts);
5. Only use software applications written by ourselves—software dictates output, we dictate software (authorship cannot be granted to those who have not authored!)

The ICA event where this manifesto was delivered marked McLean and Ward's second public performance as slub, and it occurred well before they had contacted other live coders. Indeed, although the two had explored live coding in passing at an event earlier that year,[19] on this occasion they ran preprepared scripts from the command line, rather than writing their code live. They were concerned with real-time control and with asserting their creative agency as programmers writing code to make music, but several years passed before they put these two motivations together to become an exclusively live coding collaboration. As others report, despite all the ingredients being

there, it seems that the significance of live coding, which seems obvious in retrospect, only really became apparent when communities of practice began to form around it.

The Foundry and Public Life

The growing accessibility to the internet was a significant factor through this period, allowing experimenters to make contact with communities of practice beyond their local environment. KS reports looking for dance on the internet, SA was searching for other calculator hackers, and AM was building online communities via bulletin board systems. However, the live environments of many venues and project spaces, as well as conferences and festivals, are what really provided a frame for making connections and sharing practice. The emergence of live coding was shaped not only by the evolution of the live coder as a performer but in relation to an audience that participates in the way that *thinking-in-public* is made visible through coding. Around the turn of the millennium, with the first internet boom, code was slowly entering public consciousness, as code became popularized in advertisements, on television, in magazines, and on film through design tropes such as CamelCase typography, arrays of zeros and ones as binary signifiers of the digital, and the code aesthetic of the *Matrix* movies.

The generative music scene in London gathered an audience through some remarkable venues, including the Foundry pub in Hoxton and the literally underground Public Life, an ex-public lavatory in Spitalfields curated as an experimental venue by Siraj Izhar from 2001 to 2004.[20] The Foundry pub was set up with the support of Bill Drummond, well known for his pop music disruptions as part of the KLF. Drummond's poster titled "I COULD FUCKIN' DO BETTER THAN THAT" was hung in the Foundry as an open invitation to artists to participate. Many experimental events, including the first slub performances, were held here. Public Life was run as an open arts collective with a similar focus on experimental digital culture, with the following rules of operation: "1. not to solicit activity, all activity had to be self-initiated, volunteered or uninvited" and "2. not to say no to anything, reject anything but attempt accommodation in some way." The program of events in late 2002 included Plug and Play nights, open-source occasions where participants were invited to "bring data/bring laptop/tech." Adrian Ward and Alex McLean performed as slub, while Nick Collins appeared as 3play with John Eacott and Fabrice Mogini, students working with Martin Robinson, who was teaching one of the few SuperCollider-based courses at Middlesex University at the time. Other Middlesex students organizing events at Public Life included Thor Magnusson and Enrike Hurtado, who organized events where people would use the *ixi instruments* they made as part of their master of arts degree.

Partial archives of the Public Life calendar show the determination of those in the scene not to take themselves too seriously. Variously described as "dead-eyed generative techno boffins *slub*" and "stub et de l'autre coté de slab puisque slub existe," slub and colleagues reveled in a geeky aesthetic that both acknowledged the dubious fashion status of this underground genre and exemplified the art-school habit of tongue-in-cheek "pretension" (typified by the use of manifestos). Although showing screens to the audience was central to the ethos of both Public Life and the "Generative Manifesto," it was not presumed that the performers would necessarily be writing code at the event. AM remembers that his code was generally prepared in advance and that Adrian Ward, who did write code while performing, did not make a big thing of it.

Read_me/Runme

The London scene around the Foundry and Public Life and the "Generative Manifesto" of slub and their software art prizes at Transmediale were all political in their motivations and intentions. This spirit of art activism linked up with an international community associated with the Read_me festivals curated by Olga Goriunova and Alexei Shulgin and their online incarnation at the runme.org website. After an initial event in Moscow in 2002, the second Read_me/runme was held in Helsinki in 2003. Amy Alexander and Alex McLean were invited to contribute to the selection and curation of the festival, working closely together in AM's development of the art database runme.org (with conceptual input from Pit Schultz, Florian Cramer, Matthew Fuller, the Yes Men, and Thomax Kaulmann). Runme.org became central to the Read_me festival concept as a communal catalog, submission system, and repository for executable open projects that recognized the emergent practices of coders.[21] The third iteration was in August 2004,[22] where Read_Me 2004 (hosted by Aarhus University) was immediately followed by a joint Runme/Dorkbot City Camp hosted by the Jutland Academy of Fine Arts, attended by many of the live coding community and for some (such as DG) their first experience of writing code in public. Amy Alexander, Alex McLean, Nick Collins, and Fredrik Olofsson memorably took to the stage all at the same time, with an array of projectors showing all of their screens.

The Birth of TOPLAP

The specific term *live coding* appears around the year 2000, although it must be recognized that the terms *live programming* and *live coding* were being used interchangeably at that time. It seems there was something in the air, with a number of disconnected developments—the release of early versions of the Just In Time programming library for SuperCollider in 2001 by Julian Rohrhuber, experimental programming workshops,

the first performances from collaborative ensembles such as slub and Powerbooks_UnPlugged, the release of the ChucK on-the-fly language by Ge Wang in Princeton[23]—all leading up to the establishment of the international live coding community TOPLAP at the Changing Grammars symposium convened by Renate Wieser and Julian Rohrhuber at the HfbK in February 2004.

Changing Grammars was a politically engaged festival of code-as-art, a "live audio programming symposium," whose advertised purpose was recognizably to discuss live coding as we know it today: "Some computer languages allow changing a running process on the fly by rewriting the code that defines it. Applied to computer music this means that one can write sound compositions while they are already playing."[24] The symposium brought together different forms of collaborative art. It was inspired by live improvisation in clubs, the conversational art by Antje Eske and Kurd Alsleben, network music as pursued in the work of Alberto de Campo, and a radicalization of open source as collective thinking—the name encapsulating the ambiguous power of grammatical law.

Changing Grammars was the watershed moment of live coding, as the meeting at which TOPLAP was formed and immediately after which the TOPLAP manifesto was first drafted. The (temporarily named) Temporary Organisation for the Promotion of Live Algorithm Programming was constituted, continuing as an online wiki and email discussion list that included additional software artists and live coders outside Europe, such as DG, GW, and AA. The logo, designed by Adrian Ward, symbolizes both the centrality and provisional/reversible status of the *laptop* as the name then given to an electronic music genre that was both technologized and commoditized.

The TOPLAP Manifesto

The TOPLAP manifesto itself is as "temporary" as the organization and has only ever been published as a draft, in 2004. The authors of the manifesto insist that it was never

Figure 2.2
The TOPLAP logo created by Adrian Ward.
Source: Wikipedia, "ToplapLogo," last modified April 20, 2011, https://toplap.org/wiki/ToplapLogo.

(and shouldn't be) really finished—and of course that TOPLAP is a temporary organization.[25] The manifesto was originally shared as an openly edited wiki document, settling on the following draft during the space of a year.[26] The manifesto states:

We demand:

- Give us access to the performer's mind, to the whole human instrument.
- Obscurantism is dangerous. Show us your screens.
- Programs are instruments that can change themselves.
- The program is to be transcended—Artificial language is the way.
- Code should be seen as well as heard, underlying algorithms viewed as well as their visual outcome.
- Live coding is not about tools. Algorithms are thoughts. Chainsaws are tools. That's why algorithms are sometimes harder to notice than chainsaws.

We recognize continuums of interaction and profundity, but prefer:

- Insight into algorithms
- The skillful extemporisation of algorithm as an expressive/impressive display of mental dexterity
- No backup (minidisc, DVD, safety net computer)

We acknowledge that:

- It is not necessary for a lay audience to understand the code to appreciate it, much as it is not necessary to know how to play guitar in order to appreciate watching a guitar performance.
- Live coding may be accompanied by an impressive display of manual dexterity and the glorification of the typing interface.
- Performance involves continuums of interaction, covering perhaps the scope of controls with respect to the parameter space of the artwork, or gestural content, particularly directness of expressive detail. Whilst the traditional haptic rate timing deviations of expressivity in instrumental music are not approximated in code, why repeat the past? No doubt the writing of code and expression of thought will develop its own nuances and customs.[27]

With clear influence from "The Generative Manifesto" of McLean and Ward, perhaps its most famous injunction is "Show us your screens"—a mantra of the live coding community.[28] However, even this central phenomenon of the live coding movement is contested. Some regard the showing of screens as explanatory (GW) or didactic (SA), as simply a way of sharing their way of making music (for slub), while for others the gesture is underpinned by the materialist principles of granting access to the means of production, as discussed in later chapters of this book. The founders of TOPLAP were fully aware of the irony of an improvised culture being documented and systematized: the assertive style of the manifesto is parodic and, like all manifestos, *algorithmic* in

the sense of being program-like and able to generate new sociotechnical forms. In this sense it follows a long tradition of technology-based manifestos and declarative calls to action.[29] However, our own interviews (AM with AdC) acknowledge that "even if the TOPLAP manifesto obviously is intended as tongue-in-cheek, it did have an undercurrent of real ambition."

Catalysts and Precursors

So far we have focused on the genesis of the TOPLAP live coding community between 2000 and the present day, in recognition of the rich development of music and arts practice that came out of physical and online meeting points. It is also important to recognize precursors and catalysts, where individuals or small groups engaged in activities and developments that could be conceived of as live coding even if these people never viewed their practice as such or found value in the term *live coding*.

Interactive Programming

The propagation of the Forth programming language through the 1970s was a significant event in the development of interactive programming for real-time applications, as the language architecture allows operations to be interactively (re)defined while the program is running, with a core that is unusually fast and lightweight.[30] In 1979 Doug Collinge created the Moxie language for timed procedures, delivered originally as a Forth package and later reimplemented by Roger Dannenberg in a C version called Moxc,[31] described as an "integration of procedural and declarative score-like descriptions of interactive real-time behavior."[32] Dave Anderson and Ron Kuivila began working together in 1984, initiated by Kuivila asking if it would be possible to keep access to the Forth outer interpreter while running his concurrent thread package. It is explorations in this style that have enabled live coding as we understand it now, and according to the TOPLAP wiki, Kuivila is the person responsible for the first known performance of what we would now recognize as live coding, at STEIM Amsterdam in 1985.

SuperCollider

If the formation of TOPLAP was a pivotal moment in the crystallization of live coding as a performing arts genre, the next most significant occasion (though earlier in the timeline) may well be a workshop that was presented at the International Computer Music Conference (ICMC) in 2000 by James McCartney, author of the SuperCollider music programming language.[33] McCartney's work influenced many of those we interviewed. Indeed, it is hard to ignore the number of live coders whose first experiences of coded music came from tutors who were the early adopters and teachers of

SuperCollider. Even TOPLAP owes its existence to McCartney's work; SuperCollider was the main programming language used at the Changing Grammars meeting.

Versions of SuperCollider were already in circulation before 2000, and some schools were using those early versions. But McCartney's workshop at the International Computer Music Conference 2000, where he demonstrated fluent synthesizer programming in a text-based language, made a huge impact on many of those who attended. His workshop involved a live software demonstration, not a musical performance, but many of those present recognized the potential for using code as a musical instrument, rather than as a means to build instruments.

The basic operating principles of SuperCollider are squarely in the tradition of the MUSIC-N family of digital synthesis systems, continuing Max Mathews's original invention of a modular system of software unit generators that are connected into a logical network. Conceptually, this is the same operating principle as modular analog synthesizers, which are connected into a network using patch cables, and of the Max/MSP language, which visualizes the patch cables as connecting lines in an on-screen diagram. McCartney's first version of SuperCollider provided a text programming interface to this underlying model, but Version 2 of SuperCollider transformed the programming experience by changing the style of control language to that of Smalltalk—the innovative language developed by Alan Kay and others at Xerox Palo Alto Research Center (PARC) that underlies a recent resurgence of interest in "live programming" within the software engineering research community.[34]

It is notable that Collinge's original Moxie referenced Smalltalk, and also the Simula language that had inspired it, relating the event-simulation functionality of Simula to a causal model of music in terms of processes and events, which he felt would become the foundation of a far more powerful descriptive model. Certainly, for Kuivila, SuperCollider was the first environment that he believed had retained the interactive programming quality of Forth while allowing much richer programming abstractions.

Immediately, SuperCollider 2 encouraged live experimentation. In particular, from the moment when the interpreter remained usable during synthesis, it allowed for a redefinition of functions at runtime (in the same way that words could be added at runtime to the Forth dictionary). These functions could be picked up by sequencing unit generators like Spawn, and Kuivila created a TSpawn function that introduced the ability to trigger this update. This was used in the early versions of Rohrhuber's work on proxy systems.

It is within the SuperCollider community that we see terms such as *live programming* and *code live* used around 2000–2001 by Julian Rohrhuber and Fabrice Mogini, respectively. Indeed, at the launch of the *Morpheus* CD-ROM (a disk that contained a

runtime version of SuperCollider with generative music pieces by five composers) in 2001, Mogini was live coding in a London Shoreditch nightclub, projecting code onto the wall and writing SuperCollider patterns in a specific graphical user interface system he had built.[35]

It is no coincidence that these developments were all taking place within SuperCollider 2. Its extreme conciseness, together with its repository of interesting examples that could be easily rewritten and combined in live textures, foregrounded code as public artistic expression. Its design prompted users to run different parts of their code concurrently, changing parameters continually as the music evolved. However, a considerable advance for live coding came with Rohrhuber's introduction of proxies, which made it possible to rewrite any component of the program at runtime. Instead of preparing the parameters for live interaction, the whole programming activity became an integral part of the running program. The proxy system was also used in improvisation to share and access code running on multiple computers, enabling the live laptop network music performances of the PowerBooks_UnPlugged ensemble.[36]

The Growing Community

The pioneering years of live coding in the early 2000s were followed by accelerating growth of the live coding community. It has now grown to include so many prominent individuals that it would have been impossible for us to interview them all or even to fully document the expanding circles of those involved. In the months and years immediately after the formation of TOPLAP, those who attended and participated in the early events (such as DG) developed their own tools and performance practices. Increasing coverage in forums enabled a wider network of explorers to identify their own work as live coding, as when AS saw a *Slashdot* article by AM on hacking Perl and recognized that his own "live" exploration within a read-eval-print loop could be performed in public, leading directly to the first live coding performance in Australia by Andrew Brown and Andrew Sorensen (as aa_cell), at the Australasian Computer Music Conference in June 2005.

As academic music departments started to incorporate live coding (or at least, further attention to SuperCollider) into their curriculum, they created a generation of performers who had an established framework within which to explore. AC was already familiar with computers as media-processing tools but had not considered herself to be a coder until Ernesto Romero taught a live coding class in Mexico in 2009, leading to her own first public performance in 2010. SK had already used digital music tools in high school and found that Max/MSP and SuperCollider were part of her undergraduate curriculum in 2006 and 2009, respectively. For her, coding in itself was not a

conceptual commitment but a tool available as part of the repertoire of free improvisation practice, in which she saw little real distinction between digital and analog synthesizers. She did not perform explicitly as a live coder until 2013 but quickly became a core member of the UK community.

JR and AdC were visiting professors in Karlsruhe who taught SuperCollider in 2009 to students at the Institute for Music Informatics and Musicology, including Juan A. Romero, Holger Ballweg, Patrick Borgeat, and Matthias Schneiderbanger, who then formed the live coding collective Benoit and the Mandelbrots. They say that their first performance later that year was memorable for being even "more nerdy than Kraftwerk" because they had not yet acquired laptop stands and were obliged to sit on the stage while typing.

Although more could be said about the institutional grounding of live coding through educational establishments, including the teaching of environments such as SuperCollider, Pure Data, Max/MSP, CSound, ChucK, Hydra, and TidalCycles and the widespread rollout of Sonic Pi into schools as well as universities (currently approaching its two-millionth reported download), much of the community growth and education has been through workshops organized by art institutions, media labs, universities, and various festivals. As examples, John Eacott organized the SuperCollider summer school at Westminster University in the early 2000s; Nick Collins, Fredrik Olofsson, and Sergio Luque gave SuperCollider workshops; Thor Magnusson and Enrike Hurtado organized ixi workshops on audiovisual programming across Europe; Bruno Ruviaro and Fernando Lopez-Lezcano at CCRMA at Stanford, and a large open-source Pure Data community, with people such as Derek Holzer, Gunter Geiger, and Hans Christoph Steiner, was active in open-source audio programming education. Various institutions, organizations, and collectives were instrumental in this spreading of skills—notably, Centro Multimedia, Centro Nacional de las Artes in Mexico, the Studio for Electro-Instrumental Music in the Netherlands, Access Space in Sheffield, and l'ull cec in Spain, which organized a coordinated program of SuperCollider education through a series of workshops at Hangar Barcelona.[37]

The mixture of curiosity, enthusiasm, and engagement shown by technical audiences has become a staple of live coding performance. At programming language and tool conferences, especially, there is a long-standing tradition of live demos exhibiting the convenience and efficiency of new technologies by showing that sophisticated results can be achieved in view of the eyes of the audience.[38] A second wave of live coders, such as MH and SA, extending the technical tools developed in earlier live coding experiments, were already familiar with this kind of code performance before they started to explore musical ideas of their own. For more technical audiences looking

to develop their own skills and understanding, knowing how the tools work and how effects are achieved is greatly valued, leading to performances that often consist of a lecture or seminar followed by music improvised in the moment.

Live coding has also extended beyond the performance of music and projected visual effects to explore the potential of code in association with dance or poetry. KS is a dancer and choreographer who had used digital tools such as Isadora and Processing to create dynamic stage sets and had an intuitive understanding that these were themselves a kind of choreographic notation. After meeting AM, she began to explore the intersections between long-standing questions of notation and improvisation in dance (to be discussed further in chapter 4) and the potential of live code as an improvised score.

As with any performance or musical genre, we stress the point that live coding is a community construction. Performance implies an audience; it is a fundamentally social enterprise. The tools used in live coding may be constructed in private, and live coders develop their skills with personal preparation, but experiences of community were central to the development of everybody we interviewed.

For those live coders educated in a Western art school tradition, ensemble performance and studio settings are everyday routines. Just as with the art-rock bands of the 1960s and 1970s, several prominent live coding groups developed directly from groups of art-school contemporaries, including Benoit and the Mandelbrots and the Public Life regulars 3Play. However, the cultures of education in technical practice are very different. In comparison to art schools, there is far less expectation that computer science and software-engineering students will work closely with their peers on an everyday basis. As a result, the communal experience of live coding, for these individuals, was challenging but transformative. SA discovered a completely different kind of creative experience through a classic rite of passage, forming a band (under the name Meta-eX, with his brother-in-law Jonathan Graham), while GW recalled that his undergraduate interest in algorithmic composition, to be followed by research in the Western contemporary tradition, underwent an epiphany when he had to explain his plans to band members he met at a party on the way to start his PhD in Princeton.

For a traditional music school or conservatoire context, the community of live coding may be difficult to accommodate within conventional group structures given the quite different pace and rhythm of (intermittent) edit and (continuous) execution, in contrast to the conventional instrumentalist's constant adjustment of note articulations but near inability to discuss the work while still making sound. DO did not really start live coding until 2010, with the formation of Cybernetic Orchestra. For him, the medium of code leads to forms of sharing beyond those needed for more individual musical practices. While instrumental performance gestures happen in the moment,

code gestures are temporally distanced from their effects and thus able to accommodate a parallel time stream of discourse, through which members are able to consider and grasp what others have done. Of course, code can also be an explicit technology of control, allowing those inside or outside the group to expose or impose individual intentions to a much greater extent than possible with conventional instruments.

Whether or not live coding is *intrinsically* communal, the network of personal contacts that have been described here were essential to the development of the form. Notably, AM extended the community through personal connections, such as an office across the hall from JA or meeting AC while learning SuperCollider from Romero. Once able to perform in public, the memorable venues and events of the experimental arts scene resulted in lasting impacts—two remarkable algoraves in the art vessel *MS Stubnitz* in April and May 2013; SK, BatMan, and DO visiting a tiny island in Venice for the laptop orchestra/ensemble festival Laptops Meet Musicians in July 2011; JR and RW live coding a musical accompaniment to silent movies as *Signifikantelstadl* starting in 2005; or the Hamburg-based band ginkgo turning a chill-out room into a mock office.[39]

Revising Histories, Reshaping Communities

Existing at the intersection of performing arts and computer science, live coding risks the same challenges and problems with diversity that continue to persist within these fields, as well as within technologized performance cultures more broadly.[40] Certainly, the fields of electronic art music, computing, and software engineering are not widely recognized for their diversity. However, since live coding is neither conventional software engineering nor mainstream art or music, it arguably has the capacity to emerge as a community of practice relatively free of the hierarchies and expectations that have habitually dogged its neighboring disciplinary fields.

The rhetoric of live coding's emergent community is often portrayed as inclusive, even utopian—an invitation for new collectives to form or a welcome offered to people of diverse artistic, educational, ethnic, and gender backgrounds. However, although the contour or boundary that gives shape to an emerging community creates conditions of belonging and commonality, it can also mean that there will always be individuals who do not belong or remain unrepresented.[41] While live coding aspires to be an inclusive community of practice, it is not always as simple as that. As live coder JA points out, "Computing has been coded masculine."[42] The gateway of entry might well be open in principle, but this does not always mean that its threshold is easily crossed. Networks and communities emerge through complex webs of association and initiation, friendship, and fraternity. Educational experiences can often create conditions of

expectation and convention, establishing unspoken rules and permissions, possibilities, and limitations. For example, the early educational experiences of prominent digital artists such as AA were significantly shaped by the continual socialization of women *not* to be the stereotypical coder. While the educational climate has changed, female live coders, such as AA and, more recently, ALGOBABEZ (JA and SK), have ironically appropriated the persona of the stereotypical performing nerd, celebrating the counterexpectations of adopting subversive gender identities. Increasingly for some female live coders, however, the challenge is also one of moving beyond ironic critique toward actively shaping and redefining the future of live coding in more affirmative terms.[43]

In spite of the aspiration of live coding to be inclusive and diverse, and though female coders such as AA, AC, and SK have become key figures within the live coding community, the number of male live coders arriving in the community constantly threatens to dominate. Due to the rapid increase in its size, there is a real risk that the growing community will become less diverse than even at its inception, potentially recapitulating the historical displacement of women pioneers in electroacoustic music and the sonic arts.[44] Tactics for addressing the continuing gender diversity imbalance take two distinctive approaches: the active reshaping of a community through interventionist and activist strategies and the rewriting of existing histories as a corrective toward a more inclusive narrative to reimagine a different future. Projects such as OFFAL (Orchestra for Females and Laptops) and the duo ALGOBABEZ have affirmatively foregrounded the female gender of their performers.

Significantly, many female live coders have also been instrumental in establishing specific communities and educational opportunities for supporting women's engagement in live coding, reshaping the future of live coding through advocacy and activism, through women-only workshops, and through mentoring and other initiatives to widen participation. For example, in the UK a pivotal moment for redressing gender balance within live coding was the free all-women Live Coding Workshop led by the Yorkshire Sound Women Network at Huddersfield University (December 2015) and funded by the Arts and Humanities Research Council Live Coding Research Network, which introduced twenty women to ixi lang and SuperCollider.[45] Further widening participation, JA has since led live coding workshops at the National Media Museum in Bradford, England, as part of "Make Some Noise," which targeted students (around the age of eight to nine years) from areas with low socioeconomic backgrounds. In 2019 the Northern Sound Collective hosted a series of symposiums, workshops, and hackathons (open only to women, including trans women and nonbinary people) with the theme "Automation and Me: Living an Algorithmic Life" to explore themes of bodies, technologies, and automation.[46]

In her article "Finding Joy in Error and Space to Fail In," JA argues that "in computer science pedagogy, one argument that has been put forward to explain the lack of engagement from women is their fear of failure."[47] A significant characteristic of the advocacy and education platforms described above is that they reportedly create safe spaces for failure, for taking risks and trying something out. Rather than foreground the need for technical virtuosity, such all-women and nonbinary opportunities offer, as JA argues, "underrepresented groups . . . a safe space to fail in their learning . . . a space in which to fail constructively."[48] Strikingly, JA reports that while many female live coders feel encouraged by advocacy and educational projects and even by the inclusive culture of algoraves, they continue to feel excluded from key community spaces such as the live coding Slack channel.[49] JA argues that "the dominance of technical discussion on Slack underpins the maleness of the forum,"[50] where the focus of discussion on tools and technicalities can both alienate and undermine female coders.

The introduction of algorave guidelines has also been an important gesture in changing a culture within the live coding community[51]—for example, by encouraging promoters and organizers to program lineups that reflect a greater gender balance and diversity more broadly. This is a positive intervention, yet there is still a risk that such practices inadvertently reinforce the sense of live coding as a predominantly white male culture whose rules and codes might be modified (slightly) to better allow *others* in. How can the live coding community become truly inclusive, not simply allowing or giving access to a wider diversity of individuals but rather becoming truly co-constituted and actively reshaped by the diversity of its evolving members? How can the language of inclusivity shift from one of "letting in" and "opening up," inadvertently reinforcing a sense of its own boundaries and gatekeepers, toward something that better reflects the transformative possibilities of greater diversity? In order for live coding to be a connecting force across diverse communities of practice, structuring hierarchies should continue to be rejected and shaken up when formed.

In compiling the partial histories of live coding, we have become increasingly aware of omissions and exclusions and of various biases, privileges, and blind spots, not least in relation to matters of diversity. Live coding aims to be inclusive; still, its evolution has been (and continues to be) uneven and nonlinear. While the early histories of live coding have a specifically European focus, in recent years the practice has evolved and developed into a truly global phenomenon. As live coding practices and communities develop in Africa, India, Japan, and Mexico, they are shaped and informed by the distinctiveness of those geographical, cultural, and sociopolitical contexts. The increasingly international nature of the live coding scene, particularly the strong communities that have grown in Central and South America, has begun to counteract its previously

European- and US-centric focus. For example, the /* vivo */ festival in Mexico City in 2012, the International Conference on Live Coding (ICLC) in Morelia in 2017, and the "Livecoders latinoamericanos" day programmed at the ICLC in Madrid 2019 revealed the radical vibrancy of the Latin American live coding communities to the wider scene, with the ICLC now held in both Spanish and English. While diversity in the live coding scene has been discussed and confronted from the early days, the focus has tended toward issues of gender, rather than aspects such as race and class.[52] However, events such as the police murder of George Floyd on May 25, 2020, and the impact of the Black Lives Matter protests that followed brought the matter of race into focus with a heightened sense of urgency, calling for intervention and activism beyond expressions of solidarity and performative allyship. The (Algo|Afro) Futures mentorship program is an initiative led by digital artist and curator Antonio Roberts in England that supports four Black artists in the West Midlands area who are in the early stages of their careers. The first cohort of this program—digital artist/musician Rosa Francesca, fine artist/filmmaker Emily Mulenga, mixed-media poet Samiir Saunders, and musician/illustrator Jae Tawallah—were encouraged to explore live coding from the perspective of their own practices rather than *fix* any problem in the existing scene. They approached the theme from an intersectional standpoint, bringing varied interests and backgrounds in neurodiversity, embodiment, anticapitalism, feminism, queerness, postcolonialism, and disability justice.[53]

In parallel to the interventions and activism intent on changing the existing culture of live coding, there is also potential in revising the history of live coding itself, revealing alternative narratives in which the historical presence of people who are disadvantaged and/or underacknowledged (based on gender identity, ethnicity, or class, for example) are properly recognized within the evolution of computational technologies. The trouble with histories—partial or otherwise—is that the collective memory of events and the visibility of certain individuals or practices afforded within a particular historical narrative are inescapably conditioned by the privileging norms of the social and cultural milieu itself. At times, then, a more active reading of alternative histories—of a more overtly revisionist or reparative history—becomes critically necessary for addressing those existing and persisting blind spots and exclusions, for telling *his*-story from a different perspective. For example, at the International Conference on Live Coding in Hamilton (2016), Amy Alexander's keynote "At the Margins" highlighted historical examples within computer science and the arts that were considered *marginal* or peripheral in their time but have had a long-term impact on their fields.[54] These included the seemingly clerical work of early telephone switchboard operators alongside the innovations of influential female programmers, including Rear Admiral Grace

Hopper, a pioneer of computer programming who popularized the idea of machine-independent programming languages, leading to the development of COBOL, the early business-oriented programming language.

Certainly, the role of women within the evolution of computer technology and programming generally requires ongoing reappraisal. The pivotal contribution of named individuals such as Ada Lovelace (1815–1852, known for being the first "computer programmer" writing an algorithm for execution by a computer—Babbage's Analytical Engine) or Margaret Hamilton (credited with invention of the term *software engineering*[55]) must be remembered in the context of innumerable unnamed others, such as the thousands of female "human computers" operating in the 1940s whose skill, knowledge, and dexterity have been historically undervalued. While software engineering emerged from the space program, in the military and patriarchal context of the Cold War, the golden age of electronic and computer music took place in more independent, less hierarchical, and therefore less male-dominated scenes. As it has developed, both commercial and academic power structures have formed,[56] but these do not necessarily represent the diversity of the early pioneers. Reflecting on the stellar lineup of the 2016 Mothers of Invention festival in London, Laurie Spiegel commented:

> Why so many women in early electronic music? Back then there were very few women composers of any kind. That was partly because men controlled access to performance, presentation, preservation and publication resources (concert venues, print and vinyl, concert halls, orchestra, radio airplay . . .). In contrast, electronic equipment did not treat women any differently from men. We were able to turn our musical ideas into sound and then play them for people without the almost-always-male establishment gatekeepers and their biases.[57]

There have been conscious efforts to reject such hierarchies from developing in the live coding field.[58] Beyond the reparative retelling of an existing evolutionary chronology, there are parallel attempts to radically rethink the chronology itself, to establish alternative historical references to contemporary coding that in turn invite a rethinking of its possible future.

Textiles and Coding

One attempt to revise or even rewrite the historical evolution of computing and coding, specifically in relation to live coding, can be witnessed in recent research projects that reconsider the links between coding and textiles. The development of the Jacquard loom is often credited as a key precedent in the evolution of computational hardware. First demonstrated in the early 1800s by its inventor, Joseph Marie Jacquard, the Jacquard loom served to simplify and accelerate the manufacturing of textiles using a system of punch cards to control a mechanized process of weaving. These punch cards

inspired the use of perforated cards by Charles Babbage in his Analytical Engine, and the Jacquard loom is thus considered a key technical precursor in the history of computing.

However, this focus on hardware and machinic evolution often neglects the cognitive connections between weave and code.[59] Indeed, as philosopher and cultural theorist Sadie Plant asserts, "The computer emerges out of the history of weaving. . . . The loom is the vanguard site of software development." By eschewing the well-established (accepted, normative) relation between the technologies of weaving and coding, research projects such as Weaving Codes/Coding Weaves (2014–2016) have focused more explicitly on the qualities of cognition and creativity emerging within traditional handloom use and how the handweaver's sense of embodied cognition can be extended to the practice of coding.[60] The project argues that automated Jacquard weaving is antithetical to the principles of live coding—its planning in advance seems closer to the hands-off, automated *generative music* ethos developed since the 1950s—and so to find analogs to live coding, we have to look for roots that go further back in history.

Multimedia artist and technologist Paola Torres Núñez del Prado has connected live coding with Peruvian textile practices through her *Textile Patching* performances, live coding with embroidered interfaces at the International Conference on Live Coding in 2017. More recently, she has launched the Neokhipukamayoq manifesto,[61] presenting a vision for technology grounded in a heritage based on *Khipu*-making, the pan-Andean method of digital memory storage with knots in string, known for its use by the Inca Empire for recordkeeping. In this manifesto Núñez del Prado brings forward a living history of technology that is respectful of its origins as well as its contemporary presence in Andean culture and mindful of its growing presence in the arts worldwide. The importance and indeed urgency behind this manifesto is in asserting a technological hierarchy where Indigenous thought systems are respected, where data representation is literally textile and connects with a wider vision of ecological equilibrium. Where live coding proposes a vision of technology with the programmer part of the program, the Neokhipukamayoq manifesto steps back to include culture and the environment in the loop. As stated in the manifesto:

> In these times of climate emergency, it is imperative not only to question the materials used in our practice nor their energy consumption nor their carbon footprint, but their design, symbolism and historicity. . . . Another technological narrative from the arts is imperative, one that will begin with the study of divergent technological expressions of the past and present.

From Instituting Moments to Institutional Frames

The focus on textiles (and especially ancient textile processes) traces live coding back to its earliest of precedents, conceived as an "instituting moment" for live coding practice.

However, the insurgent energy of the instituting moment or movement—the emergence of something new or novel—eventually becomes absorbed by or becomes itself an institution.[62] The TOPLAP manifesto was important in galvanizing the terminology, spreading it, and contributing to the realization that programming computers live on stage is a valuable activity. Many of the early live coders who started out as young artists and student rebels have now graduated to become academics—research fellows or tenured faculty—while others are loosely connected to academia even if working as independent artists.

In combination with the continued growth of the live coding community and its intersection with the goals of technology researchers as they design more dynamic software-engineering tools, the dynamism of this cross-disciplinary enterprise has inexorably led to research grants, conference series, and academic publications. The German computer science research center at Schloss Dagstuhl hosted a 2013 meeting dedicated to live coding for art, education, and engineering—an event at which the plan for this book was first proposed.[63] Subsequently, funding for a Live Coding Research Network from the UK Arts and Humanities Research Council, directed by Thor Magnusson and Alex McLean, supported a diverse range of events and meetings, starting with a launch event in March 2014 at which NC was invited to present a comprehensive history of live coding. Following a series of workshops and symposia, the first ICLC was organized by the Live Coding Research Network in 2015 and held at the University of Leeds.[64] A parallel series of events, whose organizers were also represented at the Dagstuhl meeting, has focused on engineering rather than the artistic implications of liveness in software development.[65]

At the same time, live coding has resisted the hierarchies of control that sometimes build around academic disciplines, and much of the creative and technological activity now takes place outside of institutions. The current wave of groundbreaking new systems, such as Hydra, Orca, Flok, Improviz, and Bespoke, have all been developed by independent practitioners, and the ICLC has increasingly promoted an atmosphere that is as much a festival as an academic event, aiming to be accessible to those without institutional backing through free or low-cost entry and travel grants. Likewise, projects born in academia, such as TidalCycles and Sonic Pi, are now thriving as independent projects sustained by their communities. In the case of TidalCycles, this is via Summer of Code grants and an OpenCollective fund, while Sonic Pi receives support from donors via Patreon. Nonetheless, academic research continues apace, research projects and studentships continue to be funded, and centers such as the Networked Imagination Laboratory at McMaster University are able to benefit from exchanges with other thriving cultures of practice, such as Colectivo de Live Coders in Argentina and

NL Coding Live in the Netherlands, among the thirty-two local communities loosely affiliated as TOPLAP *nodes*.[66] In 2021 the European Culture–funded project On-the-Fly included both artistic and research residencies in a varied program held at collaborating institutions in Barcelona, Ljubljana, and Eindhoven. As it does in many other areas, live coding challenges the accepted institutional distinctions, including that between the artist/practitioner as distinct from the academic.

Algorave

While the *instituting energies* of live coding have been absorbed and assimilated into institutional practices, and indeed while many of the first generation who are still active as live coders operate within these institutional frames, the insurgent and irreverent imperative of live coding persists. Parallel to the increasing institutionalization or even professionalization of live coding, the new genre of *algorave* has emerged. Coining the word *algorave* as the name of a new genre was a decisive moment in the history of live coding. Informed by NC's own research into the development of electronic pop music genres, AM and NC realized that a musical genre could become a rallying standard for a wider understanding of live coding.

Figure 2.3
Algorave logo designed by David Palmer, a "spirangle" based on the three-armed algorithmic structure of the Brigid's cross.
Source: Wikipedia, "Algorave," last modified September 12, 2021, https://en.wikipedia.org/wiki/Algorave.

Though the word *algorave* is undoubtedly ironic in part, just as was the case with TOPLAP, it caught the imagination of both an older generation of music and technology writers who had participated in the rave scene of the 1980s and a younger generation of nightclubbers accustomed to electronic dance music. For example, MH was among a new generation of live coders who were inspired to use their technical skills to recover earlier music ambitions (in his case, as a promising young composer before entering the software industry). The popular technology magazine *Wired* published a short profile of algorave,[67] following a key early event on the "art ship" *MS Stubnitz*, followed by articles in *Vice*, *Dazed and Confused*, and the *Wire*.[68] This media interest exposed a new audience to the phenomenon of live coding that was many times beyond what had been seen at that point. In turn, as the practice has developed so have emergent guidelines for algoraves, which while demonstrating resistance to institutions or institutionalization, as well as against hierarchies, nonetheless stipulate an underpinning ethos and set of principles.[69]

A key question for live coding is what happens when the method becomes ubiquitous, accepted, and familiar. When the novelty of the performative situation wears out and the practice infiltrates regular club events or festivals (such that there is no need to advertise special "live coding" or "algorave" events), will this mode of practice simply become part of mainstream programming, irrespective of the method of performance? Many of the live coders interviewed remain on the periphery of the technical, academic, and commercial establishment—and are concerned that growing mainstream audiences might lose the spirit of anarchy and subversion out of which the movement has grown. Indeed, Thor Magnusson has argued that although live coding is a term that has an important initial function when we introduce and explore this way of working with our machines, at some point the method will be so natural—to program a computer in live contexts—that we will not need to focus on the method anymore.[70] This transition from marginal to mainstream can be witnessed in relation to video art, media art, or, to a certain extent, even live art—specific named practices that were once clearly differentiated and invited dedicated attention and audiences, but have gradually been assimilated into contemporary art more broadly.

One way of thinking about these different arcs—from instituting moment to institutionalized practice or from interest in the medium- or method-specificity of an emerging practice to assimilation into the generic mainstream—could be through a wider consideration of group dynamics and group formation. In 1965, education psychologist Bruce Tuckman conceived a model for describing the different stages of group formation, which arguably can be seen at work within the evolution of the live coding community: forming (setting the stage), storming (meeting conflict and tension),

norming and performing (implementing and sustaining projects), outperforming (expanding the original initiative and integrating new members), and adjourning. This last adjourning (sometimes even called mourning) stage is double-edged because the expansion of an original initiative also involves breaking up or moving beyond the original group dynamic and even a sense of letting go.

New live coders—shaped by changes in technology and culture—continually add to and modify the practice of live coding and the understanding of its potential. As the practice of live coding reaches different geographical, cultural, and socioeconomic contexts, it inevitably changes. Certainly, the lived experience and articulation of live coding practice is going to vary depending on whether you are a live coder living in Europe, Japan, Mexico, Argentina, North America, Australia, or India. While the internet promises a shared global platform, in reality, practices, ideas, technologies, and theories circulate at different speeds and rhythms through different intensities and durations. The community of live coding might be described as being in its norming, performing, or even adjourning stage in one context, while in another context it is still nascent, still emerging. Indeed, our biographical interviews regularly drew attention to themes of empowerment, emancipation, and education as implications of the live coding movement. Despite the prevailing spirit of modesty and ironic detachment that has surrounded the development of live coding, initiatives such as AM's engagement with refugee-related arts and diverse community groups, or SA's fervent educational campaigning, demonstrate real engagement with the mission that has been central to the philosophy of live coding since its founding manifestos.

This book itself is pitched at an interesting moment in the history of live coding, within the arc of live coding's evolution. This chapter narrates a partial history of live coding from its emergence, through its evolution, to the point (now, as we are writing) where the stories composing this history are becoming more difficult to track and trace. TOPLAP has itself dissolved into a loose affiliation between geographical nodes with independent identities and online communities, where perhaps more discussion takes place in Spanish than English, with international activity centered around particular live coding environments.[71] In a sense, then, the book could be read as a marker or signal of a transition already taking place within live coding or perhaps (if live coding is conceived to be entering its adjourning/mourning phase) even as its obituary. Yet the complex interdisciplinary, intercultural landscape of contemporary live coding, connecting with heritage practices in order to look to the future, means that it has become increasingly problematic to view the evolution of this practice as a single linear arc. Not only is the live coding landscape always changing; its community spans multiple intersecting landscapes with a commitment to keeping ideas alive and changing that

itself disrupts and dissolves hierarchy. Additionally, the tension between disciplinary discourses—for example, between art and engineering, as we discuss in chapter 7—has been perennial in every field of technical innovation, with craft, design, and indeed music often to be found in the contested intersection. It therefore seems likely that these alternative perspectives and communities will continue to grow into the foreseeable future.

Eighteen years on from the critical gathering of the Changing Grammars symposium of 2004 and the drafting of the TOPLAP manifesto, this book hinges on a moment in live coding's evolution when it is just about still possible to narrate a partial history, however fallible, at a critical juncture from where the practice of live coding now diverges and diversifies in ways we cannot yet tell. While this chapter has attempted to give some kind of coherence or contour to the arc of live coding's development thus far, chapter 3 presents live coding in its singularity, exposing the specific approaches through which live coding manifests within a diversity of different practices. Neither comprehensive nor exhaustive,[72] these expositions are conceived as personal perspectives on the practicing of live coding *by* live coders.[73] The chapter includes both the creators of new live coding tools and those who have developed artistic practices through using those tools. Some identify primarily as technologists, some as composers, and some as performing artists. As editors of this chapter, we do not attempt to synthesize a single narrative. Rather, we provide an opportunity for the diversity of live coding to be represented through rich accounts of practice and allow the practice to speak for itself and on its own terms.

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

This book was set in Stone Serif and Stone Sans by Westchester Publishing Services.

Library of Congress Cataloging-in-Publication Data

Names: Blackwell, Alan F., author. | Cocker, Emma, author. | Cox, Geoff, author. | McLean, Alex, 1975– author. | Magnusson, Thor, author.
Title: Live coding : a user's manual / Alan F. Blackwell, Emma Cocker, Geoff Cox, Alex McLean, and Thor Magnusson.
Description: Cambridge, Massachusetts : The MIT Press, [2022] | Series: Software studies | Includes bibliographical references and index.
Identifiers: LCCN 2022008717 (print) | LCCN 2022008718 (ebook) | ISBN 9780262544818 (paperback) | ISBN 9780262372626 (epub) | ISBN 9780262372633 (pdf)
Subjects: LCSH: Computer programming—Philosophy. | Agile software development. | Creation (Literary, artistic, etc.) | Algorithms—Psychological aspects.
Classification: LCC QA76.6 .B5794 2022 (print) | LCC QA76.6 (ebook) | DDC 005.1301—dc23/eng/20220527
LC record available at https://lccn.loc.gov/2022008717
LC ebook record available at https://lccn.loc.gov/2022008718