6 Time Criticality in Live Coding

Although *time critical* is a term used in computer engineering,[1] this chapter explores how live coding presents a direct challenge to the conventional understanding of the contemporary experience of time in the digital domain. On the one hand, the programmer-subject has often been too absent from accounts of digital experience, dwelling in an engineering dreamtime, a metalevel of narratives prior, beyond, and above the experiences of the user. On the other hand, the computer itself also manipulates time, as the system clock and program counter play out the sequential logic of the Turing machine's tape. In live coding, these time regimes are coupled, human and machine times become entangled, and conceptual configurations are situated in the moment of their interpretation, subverting the mechanical determinism of the clock and introducing other temporal registers. In what follows in this chapter, live coding can be understood to operate both *in* and *out* of what we conventionally understand as time, expressing different rhythms, experiences, and epistemological registers and exemplifying the "coming together" of multiple and different temporalities.[2]

In the first section of the chapter, we introduce temporal complexity and begin to explore how live coding practices manipulate these processes of temporalization with a number of concrete examples. Following this, we further discuss the disjunction between algorithmic or machine time and the performer's embodied experience of lived time and, of course, how live coding operates across these objective and subjective registers as a practice that appears—anachronistically—both in time and out of time. The elasticity here is important because it points to disjunctions that challenge any received notion of a unified present. Live coding is thus to be understood as a complex multitemporal human and more-than-human experience. The time criticality we speak of points to this inherent processuality of the computer and programming to express dynamic operations: it is both critical on a technical level and on a conceptual level and moves toward "criticality" in the sense that we place emphasis on practices that actualize potentialities in real time.[3]

Live coding is clearly a practice of time: it takes place in time and appears to be of its time, both *extemporary* (prepared, but somewhat improvised) and *contemporary* (as if occurring in the present). It seems commonsensical to say that live coding operates in and expresses the present, but we might seek more detail on how live coding enacts a very particular sense of the "present present."[4] The work of philosopher Peter Osborne informs our initial understanding of the complexities here and provides some clarity in distinguishing not a single present but a multiplicity of complex presents. In his words this increasing temporal complexity expresses the

> distinctive conceptual grammar of con-temporaneity, a coming together not simply "in" time, but "of" times: we do not just live or exist together "in time" with our contemporaries—as if time itself is indifferent to this existing together—but rather the present is increasingly characterized by a coming together of different but equally "present" temporalities or "times," a temporal unity in disjunction, or a disjunctive unity of present times.[5]

The passage describes the more complex and layered problem of different kinds of time existing simultaneously across different spaces, as part of really existing global capitalism in which real-time technologies play an increasingly significant role in reproducing our experience of time. Osborne would insist that we are increasingly subject to the conditions of a *global contemporaneity*, underpinned by more general economic and sociotechnological processes, which makes contemporary practices such as live coding possible at all and, we would add, that makes live coding appear symptomatic of temporal complexity as well as a means through which it might be better understood as disjunctive.

When the philosopher Giorgio Agamben asked, "What is the contemporary?," he was referring to the contemporary not as mere coming together in time but as the "untimely."[6] In other words, he interpreted the contemporary as an experience of disconnection, or a disjunctive relationship with our own time, and as such a precondition for being able to act upon it and against it.[7] We speculate that live coding might similarly offer insight into our disjunctive experience of time in which humans and machines run in—and out of—synchronous time and where temporal complexity is actualized and made available to experience. Furthermore, if politics necessarily involves struggles over the experience of time, as Osborne suggests, then further discussion becomes important in developing a critical technical practice that negotiates and expands our ways of being in time, of understanding it more fully, and of even being able to somewhat shape it.

Building on chapter 5, the discussion of so-called real time provides a concrete example of the coming together of different temporal registers and encapsulates the demand for ever-faster transactions and instantaneous feedback. Real-time computation

underpins this cultural logic, as well as the wider applications of just-in-time economic production, yet there is little attention paid to this concept in art history or cultural criticism—aside from the *real-time systems aesthetics* of Jack Burnham from the late 1960s perhaps—to understanding how real time influences aesthetic practices outside of the register of liveness or performativity.[8] In changing rules at runtime, live coding seems to actualize these real-time dynamics. Put simply, the phrase *real time* refers to the effect of information being delivered seemingly as it arises. In computing it serves to describe the actual time elapsed in the performance of a computation by a computer, where the operation appears to be immediate and able to correspond instantaneously to the distinct operations of an external process—as, for example, with the fluctuations of financial markets, where accuracy and predictability is critical. But there is no such thing as *real* time to the computer, only degrees of delay.[9] This is a central discussion in terms of the emergent aesthetics and potential to offer time criticality, in which the term *criticality* offers technical and epistemic insights. We follow writer and theorist Irit Rogoff on this point, in her emphasis on criticality to challenge "the underlying assumptions that might allow something to appear as a convincing logic" and move to "an emphasis on the present, of living out a situation, of understanding culture as a series of effects rather than of causes, of the possibilities of actualising some of its potential rather than revealing its faults."[10] We would like to argue that live coding is deeply implicated in these operational dynamics in ways that reflect the intensity and disjunctive experience of time more widely.

Representing Time

In chapter 4 we noted that different live coding languages embody different technical and indeed philosophical understandings of the creators of those systems, and this is particularly true of temporal relationships between coder, code, and outcome. Live coding language developers themselves have often reflected on the nature of time within their computational formalisms, as well as technical documentation and the languages themselves to compare the different representations of time at play.

One approach to live coding language is *agent based*; some process unfolds over time, which the live coder guides by making declarative statements to create "agents" that tell them what to do. This is a social view of time, focused on the actions and interactions of agents within a space. An example is Al-Jazari by Dave Griffiths, one of several experimental gamelike live coding systems he created using his Fluxus game engine.[11] The live coder/gamer issues instructions as discrete steps, including movement and sensing, in this case to live code music as a form of cellular automata. A more overt

likeness to classic cellular automata systems is the Orca system introduced in chapter 4, where the live coder works with agents moving around a two-dimensional array of instructions. Another instructional approach is Magnusson's Threnoscope, where statements create tones moving around a surround-sound multispeaker array. One interesting aspect of the Threnoscope is the extent to which the visualization becomes part of the music, allowing the audience to anticipate sound events at very long timescales so that extended periods of silence can be incorporated into a performance as an exemplar of *slow coding* (which we return to later in this chapter). Here the coder takes an external view of time, taking a godlike position of watching agents work while telling them what to do.

In conventional imperative computer programming, an algorithm is specified in terms of step-by-step sequences of instructions organized into loops, conditional *if* statements, and named procedures. These procedures are often grouped together as *classes* that are then associated with data sets as *objects*, but within procedures, the logic of time has largely stayed the same since the introduction of ALGOL in the late 1950s. In such systems, time is not always considered to be part of the logic of a program but is always present, as a step within a sequence always takes time to process. Being able to decompose a given problem into conditional statements and loops is therefore seen as core to computational thinking in the computer science education agenda. This can be seen in the venerable and still very widely used Scratch system, which supports live coding and where word blocks can be arranged into loops and conditions using a mouse.[12]

The same structures of loops and conditionals can be seen in Aaron's Sonic Pi live coding system first developed in 2012 in a successful effort to meet the resurgent computer science curriculum in UK schools at that time. Interestingly, though, Sonic Pi takes these abstract, imperative structures and makes them concrete and declarative. This is done through the introduction of accurate timing, where *sleep* instructions are added as imperative steps, marking out time and rhythm in the resulting music. Therefore, what matters in a Sonic Pi procedure is not *what* it outputs (as with imperative programming), but exactly *when* and *how* that outcome unfolds. Rather than using loop constructions to solve problems through iteration, a Sonic Pi programmer uses them to directly represent a *musical* loop. This exposes an interesting aspect of live coding—it appears to be a form of software engineering, but on closer examination, the practices are very different.

The 2011 Extempore system and its 2005 precursor Impromptu, both by Andrew Sorensen, are similar in that music is declaratively represented, using step-by-step structures associated with imperative programming.[13] A core feature of Extempore is *temporal recursion*, in which a function calls itself with a particular time delay. This uses

what is known as *tail recursion*, in which the final step in a function is the recursive call to itself. Temporal recursion is Extempore's solution to a problem that every live coding environment has to face: How can a program be interpreted while it is itself undergoing change? A temporally recursive function always runs to the end, but if it is replaced in the meantime, it then recursively calls this new version of itself. Temporal recursion therefore manages both change and time.

That change needs to be managed in such a way follows from a fundamental *incommensurability* lying at the heart of live coding, as media theorist and live coder Julian Rohrhuber has pointed out.[14] This is the sense that the ideal of live coding—changing rules as they are followed—is in some sense impossible. Rules are followed over time, which implies that a process is dependent on its history, and it is a matter of causality that it is not possible to change what has already happened. In practical terms, in order to respond to code changes compromises must then be made in the design of a live coding environment: Does it attempt to continue with the state left behind by the previous version of the code, or does it recalculate the history of the process as though the code had always been as it is now? Rohrhuber refers to these as the *state* and *causal* pictures, respectively, as complementary but incommensurable approaches to change.

The aforementioned Extempore and Sonic Pi systems maintain a state picture, but several systems apply an alternative, causal picture to live coding system design by taking a *pure functional* approach to the representation of time. This is particularly true of live coding systems for sound synthesis, such as, to a large extent, the SuperCollider sound engine, but is also true of TidalCycles, which is oriented around pattern. The problem with a causal picture is that changes to code cause an immediate switch from one version of history to another, creating a discontinuity, which in the case of sound synthesis results in an audible *click*. One workaround for this problem is to run both the previous and new versions of the code at the same time, adding an audio cross-fade between them. This approach, taken by Rohrhuber's Just-In-Time library for SuperCollider, works very well but has a clear aesthetic impact on the musical results, especially when interferences between the old and new create a momentary chord or polyrhythm during the transition. Alternatively, some live coders switch this feature off and embrace the discontinuous click as an aesthetic choice.

TidalCycles instead maintains a causal picture and is unusual for representing pattern as a pure function of time, as opposed to working with stateful operations over sequences or lists. *Pure* here is a technical term for describing a function that is not able to *do* anything apart from calculate an output based on its input. From this tight constraint comes great flexibility; because a TidalCyles pattern has no *side effects*, its timeline is untethered and may be manipulated freely—for example, reversed, chopped

up, rearranged, shifted, and compressed—or indeed manipulated in many combinations and at multiple scales.

Olivia Jack's Hydra live coding system applies a similar causal picture in synthesizing video rather than sonic patterns,[15] specifying GLSL shaders as a function that takes time and pixel position as input and returns a color for that pixel as output. In other words, the main renderer is a pure function that has no internal representation of time, aside from taking time as input. The live coder may then build their own representation of time to work with—for example, by working with sequences over time or with oscillators with time as input. What breaks this purity is Hydra's ability to take itself as an input source, allowing video feedback effects similar to the classic analog video synthesizers that inspired Jack to create Hydra.

With all these systems, the live coder might have the impression that the program is unfolding over *physical time*. However, if that were true, then all the time-consuming operations of interpreting and executing code would be perceivable in the music as delay, continuously varying and resulting in off-kilter stutters. With few exceptions,[16] live coding systems therefore differentiate *logical* and *physical* time in order to achieve metronomic accuracy. Calculations then work ahead of time, working out when things *should* happen and then introducing delays so they happen at exactly the right moment.[17] These calculations must be continually adjusted to stay accurate with physical time, as measured against a system clock. The temporal reference point for physical time is itself physically manifest, in the form of an oscillating crystal located within the circuitry of a CPU or sound module. Although computer programmers work many levels of abstraction away from these crystals, they are still required to keep the whole computer in coordinated self-oscillation so that it can usefully function. Here we can take a moment to consider the vast sociotechnical enterprise of a whole operating system, consisting of the encoded thoughts of many thousands of systems programmers working together in networked, coordinated sympathy with this shared clock.

In Time/s

The above examples from live coding inform our understanding of the contemporary experience of time, marked by its temporal complexity, multiplicity, elasticity, fragmentation, and unevenness. With its questions of latency, just-in-time programming, and collaboration in time, real-time media makes the contemporary experience of time ever more complex in ways different from the past. However, before expanding further on the implications of real-time media on the experience of time, we offer some reflections on how the human experience of temporality has *always* been shaped by different

and often competing—even contradictory—rhythms, regulations, and philosophical rhetoric.

Time is "out of joint," as Hamlet once announced.[18] Indeed, for cultural geographers Jon May and Nigel Thrift, our sense of time—especially *social time*—has been influenced by numerous factors, including the consideration of natural timetables and rhythms, through social rituals, through our relationship to various instruments and devices that mark the passage of time, and through new conceptualizations of time itself. Consequently, they argue that social time is not—and perhaps never has been—uniform or singular but rather is marked by a "radical unevenness in . . . nature and quality" in which "the (already partial and uneven) networks that constitute one domain connect (or fail to connect) with the (partial and uneven) networks constituting another."[19] Likewise, and although rather anthropocentric in tone, performance theory would tend to assert that time is "not a given, natural, objective phenomenon, but a condition and product of processes of human activity" involving "processes of temporalization . . . of perception, measure, experience and worlding."[20]

Yet, while this uneven, heterogeneous experience of time, of temporality, and even of the various processes of temporalization might feel tangible at the level of lived reality, the common means through which we articulate time—specifically through the standardized scientific model of machine time or clock time—does not account for these vagaries or for how machines might produce their own sense of time outside of human experience. So how might the practice of live coding contribute toward new understanding about our contemporary temporal experience?[21] How might live coding reveal and reflect the heterogeneous and contradictory temporalities that we negotiate within our daily lives? Moreover, how might live coding itself temporalize and actively produce—activate, intervene in, invent, and reorganize—as much as *reflect* different experiential temporalities? What different modalities of time, timing, and temporality does live coding give rise to?

In one sense, of course, live coding is a practice that actively works with temporalizing technologies: it harnesses the algorithmic logic of computer programming using those computational technologies, digital devices, and instruments so inseparable from the contemporary networked conditions of global capitalism. That temporalizing technologies radically (re)shape human temporal experience can be seen in relation to social practices that have attempted to standardize our understanding of time as, for example, the clock in relation to the process of industrialization and the regulation of the labor force. The claim can be extended to the way that contemporary digital technologies—through their privileging of velocity, speed, and immediacy—serve to both *accelerate* and simultaneously *compress* our experience of time (and space).

Arguably, one consequence of recent technological developments is the way in which those different means through which social time has been historically organized have been collapsed, or rather subsumed by, a rather more precarious temporality, a contemporary experience that sociologist Zygmunt Bauman has called *liquid times*. Liquid times refers to the "passage from 'solid' to 'liquid' modernity,"[22] the shift from a sense of a *chrono-normative* existence in which social forms and institutions provide a specific stable framework for the organization of human life toward an increased and somewhat pervasive sense of temporal uncertainty and instability.[23]

For art critic Amelia Groom, "The delocalization and non-fixity of networked digital space is both a symptom and catalyst of the broken, multifarious time that we find ourselves in."[24] Technological progress (under the influence of modernism and in turn neoliberalism) has given rise to a culture of twenty-four seven access and availability,[25] a work-life indeterminacy that exposes us to a heightened pressure to perform and a state of perpetual readiness supported by an ethos of just-in-time productivity.[26] In this sense, live coding could be seen as an exemplar of our contemporary liquid times, a precarious practice that utilizes the temporalizing technologies of our algorithmic age toward the generation of on-the-fly performance work. Here, live coding could be conceived as entirely *in time* with the conditions of its own emergence: the perfect product *of its time*. Its performativity could be seen as one of making tangible or explicit the temporal conditions of our "liquid life,"[27] at worst simply replicating, representing, and even reifying that which Capital already knows to exist.[28] However, while contemporary life might indeed appear to be organized (perhaps even disorganized) through the specifically algorithmic temporalizing technologies of neoliberalism, other temporal modalities still remain as a somewhat stubborn palimpsest. Perhaps, then, it is this temporal palimpsest (or disjunction), which Capital seeks to conceal or deny, that the practice of live coding reveals.

In the enactment of the coming together of different times, live coding enables the performer to exhibit some sense of agency over these processes of temporalization to generate and execute a specific output (sound, visual image, or movement) through algorithmic manipulation across these layers. Rhythms can be transformed at multiple scales (beyond human capabilities alone) through the modification of functions and parameters in real time. When generating an output, the live coder is required to understand these different registers of time—how code execution unfolds in time as well as how the live intervention of the coder might change its course—which all require sensitivity to timing. Specifically, the live coder has to decide *when* as much as *how* to intervene in the performance. The success—and indeed risk—within a live coding performance thus relates to degrees of control over these processes of temporalization, in particular the

timely organization of edits into transitions and shifts and into more stable sections of focused development of form (such as a rhythmic, melodic, or timbral theme) emerging from an algorithmic combination of functions or transformative steps.

The use of clock time or measured time remains an important part of this. It functions as a way of structuring time in performance practices, as choreographer and performance scholar Jodie McNeilly states: "Time used as a system of counts (time-as-timing) is a useful tool for structuring choices in the construction and composition. . . . Time-as-timing sets rhythm and pace and provides a precise measure for the structural segmentation of a performance event."[29] Indeed, some live coders perform using the time signatures and temporal constraints that are familiar within more conventional (musical) performance composition, counting as a means to decide when to shift key or change tempo. For example, when live coding in an algorave context (discussed in other chapters), the coder might draw explicitly on the signature structures of dance music, an underlying or grounding rhythm based on powers of two (e.g., four, eight, sixteen, thirty-two, or sixty-four beats). Significantly, the performer must *feel* the relation of the surface rhythms of musical performance to the underpinning pulse of beat or meter. Indeed, while computer time is metric, rhythm is something perceived rather than notated in language or written down.[30] Here, some live coders report working more intuitively, deciding to make a change or shift based on time and timing felt, rather than necessarily counted. In these terms, the code itself does not express the whole experience of live coding (and indeed this question of what cannot be articulated through code language—including the live act of listening and responding—is explored in the previous chapters on liveness and notation). Live coding performance emerges through a negotiation between the prescriptive *what* of the notation and the experiential *how* of the passage of time.

In order to effect a change at the next metric cycle, the coder has to pace their speed of typing according to this metric deadline, and if this opportunity is missed, then they might well wait until the next metric cycle to initiate the change (or for the effects to take place). However, once an algorithm is set to run it does take time: time is thus a material part of the process, with the running time of the algorithm itself structured into the live coding performance. Live coding thus involves the play of different temporal rhythms: this process involves live negotiation between the coder's desire for a change or transition and the time that typing takes (with its potential for mistakes and errors), alongside the time of execution (with its technical latency and buffering).[31] The coder has to gauge how much time is enough—the optimal time needed for both human and nonhuman operations so as not to interfere with the flow of rhythm—giving it enough time so that you don't hear how much it takes. Some live coding

systems are designed such that the delay of execution is minimal, perhaps impossible, to perceive. In others it may be longer, a delay that the live coder internalizes in each act of preparation in anticipation of the next change. Such technical delays are directly comparable to playing other instruments, such as that experienced by a church organist when a pipe takes time to sound.

In most cases, and under normal conditions, the time that it takes for simple procedural code to be parsed and executed by a computer is minimal, and the timeline of an algorithm running is practically identical to the timeline in which its immediate effects can be heard. However, this execution time does have the capacity to be creatively exploited. For some live coders, the deliberate overloading of the computer can be deployed critically, exposing the inherent rhythms of machine processing by pushing it to its limits.[32] Execution takes longer under these pressured conditions, creating glitches or pops as the computer switches between tasks, missing samples in the absence of adequate time for generating enough audio. As the computer misses deadlines, you start to hear the time that the processing is taking, giving tangibility to a largely imperceptible (operational) level of computational infra-activity. It is not uncommon for live coders to use this to finish a performance, adding more and more layers of processing until the software, and therefore the music, stutters to a halt.

Time Felt and Temporal Elasticity

Within live coding, algorithmic time is blurred with both clock time and musical-rhythmic timing. In the performance of live coding, the unfolding of algorithmic or machine time meets with the performer's embodied experience of lived time, and the seemingly objective temporality of measured time—the beats per minute of clock time—is shaped and modified according to the *felt time* of a more subjective register.

Live coding operates at the interstices between what might be described as objective and subjective temporalities, with the potential to contribute new knowledge with regard to "the problem of time experience and time perception (how we experience time passing co-relative to measured time)."[33] The discrepancy or seeming incompatibility between these two approaches to time, temporality, and temporalization—between "the irreconcilability of the apparent elasticities in the experience of time and the rigidity of its objective measure"[34]—has long been a preoccupation within both philosophical and artistic thinking. Feminist scholar Elizabeth Grosz identifies various thinkers (such as Friedrich Nietzsche, Henri Bergson, and Gilles Deleuze), alongside "philosophers of becoming" (among them Martin Heidegger, Maurice Merleau-Ponty, Jacques Derrida, Michel Foucault, Pierre Klossowski, and Luce Irigaray), whose respective writing

has attempted to challenge some of the temporal "assumptions provided by everyday and scientific concepts of directionality, progress, development, accumulation and lineage."[35] Or, more specifically, for Grosz these various thinkers can be united through their affirmation of "time as an open-ended and fundamentally active force—a materializing if not material force—whose movements and operations have an inherent element of surprise, unpredictability, or newness."[36]

In the context of performance studies, too, what underpins the attempt to conceptualize a more embodied, affective, experiential understanding of time is a general sense of "suspicion of abstracted, measured, objective time" in which the aim is to develop a "multi-temporal understanding of time grounded in human experience, perception and performance."[37] Central to this endeavor is a critique of, or challenge to, the dominance of deterministic clock time as the means through which time and temporality are commonly understood (in Western culture at least). For Grosz, "What is significant about clock time is that it homogenizes and measures all modes of passing insensitively, with no reference to or respect for the particularity of duration of events and processes. It imposes rather than extracts unity and wholeness through homogenization and reduction."[38]

In this section we also address this balance by briefly highlighting *different* models for conceptualizing lived time (extending the concerns of the chapter 5 on liveness), in which temporality is experienced through its duration rather than its spatial abstraction and the potential elasticity or stretchiness within temporal experience becomes foregrounded. However, it should be clear that in addition to these aspects, we consider nonhuman elements such as machines and code—although not the focus here—to be crucially important for an articulation of time criticality, and we return to this later in the chapter. Live coding invites reflection on the lived and embodied aspects of computer programming while encouraging an understanding of time and temporality beyond anthropocentrism.

While a comprehensive exposition on the different philosophical standpoints of *lived time* is beyond the scope of this book, some elaboration is useful in the context of live coding. One, and admittedly limited, approach to this issue is through a Western tradition of philosophical thinking drawing on the work of, for example, Bergson, Husserl, and Heidegger. Writing at the turn of the twentieth century—a period marked by profound shifts toward the automation and standardization of lived experience—Bergson's philosophy of *pure duration* presents an important attempt to think of time in ways other than through the abstract spatialization of clock time. Against the logic of mathematical time—a measurable duration divided into a sequence of distinct, discontinuous instants or units—Bergson proposes the idea of pure time or even real time, the inner experience of *time felt* in its ceaseless and indivisible continuity, as incessant flux.

For Bergson, the present moment does not refer to "a mathematic instant" but rather "the real, concrete, live present . . . occupies a duration."[39] He argues that the duration of the present moment "has one foot in my past and another in my future. . . . The psychical state then, that I call 'my present,' must be both a perception of the immediate past and a determination of the immediate future."[40] Here, as Grosz clarifies: "Time is a mode of stretching, protraction, which provides the very conditions of becoming. . . . Time is the hiccoughing that expands itself, encompassing past and present into a kind of simultaneity."[41]

This elasticity of the present moment is also explored in the writing of Husserl, whose phenomenology of lived time is articulated through a three-part present: "A present-of-the-present moment (not so different from the present instance of *chronos*, the passing point of moving time)" (impression), a "past-of-the-present moment" (retention), and a "future-of-the-present moment" (protention).[42] Within this conceptual model, as psychologist Daniel Stern asserts, the "future-of-the-present moment is part of the experience of the felt present moment because its foreshadow, even if vague, is acting at the present instant to give directionality and, at times, a sense of what is about to unfold."[43] In the philosophy of Heidegger, too, time is conceived beyond the model of *common time* or *vulgar time* (measurable by clocks, for instance) and even *scientific time* (associated with mathematics and physics, or even computer science). In advance of this model of common time, Heidegger proposes a state of original temporality (*ekstasis*) as the basic structure of presence or being there (Dasein),[44] conceptualized through a unity of three temporal phases that significantly do not refer to a linear chronological sequence of past (no longer), present (now), and future (not yet). As Heidegger states, "Temporalizing does not mean a 'succession' of the ecstasies. The future is *not later* than the having-been, and the having-been is *not earlier* than the present. Temporality temporalizes itself as a future that makes present, in the process of having been."[45] Rather, as McNeilly observes, "Past and futural events following this structure are inextricable from the present; both the having-been-ness and not-yet-now are always expressed in the now of enpresenting."[46]

This cursory glance at philosophies of the lived present is intended primarily to point toward a different temporality beyond that of common time toward a sense of lived time's inherent elasticity. Without going deeper into a phenomenology of time as such (which is beyond our scope here), our intention is to indicate how live coding also challenges the common understandings of time in ways that are experientially tangible in the encounter with the practice itself. Live coding is not so much performed in the singular temporal present; rather, it is an experimental practice capable of extending, expanding, and even stretching the sense of how the present is experienced. Live coding's temporality involves more than the mathematical time of computation.

It unfolds through the lived present and pure duration experienced by the living, embodied performer *and* the audience alike and is especially evident in the case of musical performance in the present. Indeed, the phenomenologist Alfred Schutz has defined music as "sharing of the other's flux of experiences in inner time, this living through a vivid present in common."[47] However, in the case of live coding, unlike some conventional forms of musical performance, the live actions of the performer are not entirely synchronous with what is heard; the relation between the performer's action and output might well be asynchronous (or indeed disjunctive).

The present moment of live coding's performance involves the felt *being in time* with the live, unfolding flux of a sensorial experience (whether sound, image, or movement) at the same time as the writing—or witnessing the writing—of the code itself occurs. This is an action performed in the present but future oriented, conceived in relation to a present moment yet to come. Within the live and present moment of coding, the coder is both attending to and improvising in the moment (an experience perhaps synonymous with the flow states discussed in chapter 5) while also setting up the conditions for what will come next once the code is executed. Moreover, live coders are often engaged in the production of preparation or scaffolding, code written in the present that will be activated at a future point. This preparation is what takes the most time, whereas its later activation takes an instant. In the context of programming languages, programs promise to do something—for instance, when one part of the program communicates to another. This is one of numerous mechanisms for coordinating between parallel threads or processes. In the case of live coding, we might speculate further that the promise becomes a part of the temporal relationship between the algorithm, performer, and audience.[48]

Following this it could be argued that live coding performances give expression to the promise of the three-part present, where the performer's attention appears split between the present-of-the-present moment (the *as is*) and a future-of-the-present moment (the *yet to come*). Moreover, while improvising in the present-present and planning for a future-present, the coder is also attentive to the past-of-the-present moment through backtracking and reactivating code lines already written but presently inactive. Indeed, while the live coding audience experiences the durational lived present of an unfolding performance (in time), the performer seemingly occupies an altogether more ambiguous time zone, attending in time to something as it is happening live in the continuous present while also working *out of time* or *ahead of time* on code that may or may not be executed at some point in the future. For example, the visible frame of the live coding performance—the projected screen—has the capacity to show both the temporal continuum of the working timeline—the running command

line of the executable program—and code material that is either not yet or else no longer part of that timeline. The live coder might be working on something that exists outside of the timeline of the unfolding sound to be introduced at some later point in the performance. To give an example of a specific system, Dave Griffiths's Scheme Bricks allows the live coder to drag, drop, and plug together programs rather than typing. Unwanted sections are pulled out of the program and set aside rather than being deleted, accumulating around the program as spare parts that are often later recycled by being pulled back into another section. Here, unwanted sections of code have the capacity to be pulled out of the timeline as well as pulled back in. Within different live coding systems, fragments of code can be prepared in advance in order to then be built upon or modified during the performance. The timeline of execution has a sense of temporal continuity, yet this can be stretched or contracted through the addition or removal of discrete code sequences or even code events that have their own temporal structure independent of the timeline.[49]

Different live coding systems contain different solutions as to how musical events are scheduled in time, often offering original solutions. For example, within Thor Magnusson's ixi lang system alphabetical characters are used to represent events (either numbers for notes or letters for sounds), and the spaces between them are noneventful or silent. The code thus gets a graphic, spatial, and score-like representation of the music. The temporality of code is represented by its color. Grayed-out code is not active, active code is green, yellow code is being executed, and when code is rewriting itself, it flashes white. The ixi lang typically encourages the user to work with beats, polyrhythms, and agents of varied tempi. However, a conscious effort has been made to lessen this event-based focus of ixi lang—for example, with the concrete mode and the morphing of samples in the rhythmic mode. Alternatively, Magnusson's Threnoscope system emphasizes notes of indefinite length (drones), harmonic waveforms that can be instantiated anywhere on the system's circular interface, where waveforms move in cyclical space, crossing lines that represent speaker locations in space and whose parameters can be controlled through a textual interface. Refraining from a linear representation of time, the work is underpinned by a circular temporal structure without a clearly definable beginning or end, invoking a sense of timelessness, even the eternal.[50] McLean's TidalCycles system allows both periodic and linear patterns to be represented, allowing for the temporal duality of having repeating rhythmic structures that nonetheless progress linearly. Time can be conceptualized either as linear change with forward order of succession or as a repeating cycle where the end is also the beginning of the next repetition.

In these terms, live coding systems such as TidalCycles and Threnoscope invite a consideration of time criticality beyond a Western-centric perspective—whether

philosophical or scientific—embracing the influence of non-Western thinking including, for instance, Indian conceptualizations of time and its relation to music. Here, as ethnomusicologist Martin Clayton observes, Indian cyclical time is not the "philosophy of sheer recurrence" with cycles of events that repeat themselves exactly. Clayton argues that Indian music "unfolds in a process of continuous development and does not repeat exactly, but this development takes place largely within the context of a cyclically repeating temporal structure—the tāl."[51]

An Indian conception of time can be seen most clearly in the venerable Bol Processor system for algorithmic music, created by computer scientist Bernard Bel from work on notating tabla rhythms and developed over forty years. Drawing from Indian classical music, it includes an expressive approach to time setting that seems unique to the algorithmic music field,[52] in which sound events are organized in terms of interrelationships before being mapped to physical time. Although not a live coding system itself, it has been heavily influential on the design of the TidalCycles system, particularly its embedded "mininotation" language for describing rhythm in the Bo Processor and more generally its representation of music based not on the duration of events (as in staff notation) but on the duration of cycles.

This focus on cycles rather than notes is in keeping with the longer-term focus on structure over events in live coding. In a cyclic structure, the live coder is placed in a situation in which the past is also the future, inviting a trancelike state, again comparable with the flow states discussed earlier. However, there is rarely just one cycle at play, giving the possibility of multiple cycles running at different speeds or at the same speed at different lengths, creating an interference pattern between elements, also known as polyrhythm and polymeter. A particular code state therefore may or may not produce an outcome that repeats before the next edit is executed, and the loop transforms into the next unfolding state.

Live coding is thus performed as a complex multitemporal, more-than-human experience where time can be revealed not only as a continuous, indivisible flux or flows but also as discontinuous breaks, enabling the possibility of temporal zones seemingly outside the continuum of an unfolding present and thus exposing time's inherent elasticity.

Out-of-Timeness

Live coding offers an intriguing model for reflecting on these various temporal conceptualizations since it is a practice that appears to be simultaneously in and out of time, occupying a disjunctive between space that exposes different temporal zones

of experience. Indeed, and following Agamben, we might take this to be a precondition for perceiving one's own time through anachrony.[53]

Some further elaboration of the untimely is developed in this section. Drawing on the ethnographic work of Arnold Van Gennep, anthropologist Victor Turner elaborates on the *liminal* quality of *out-of-timeness* as "an interval, however brief, of margin or *limen*, when the past is momentarily negated, suspended, or abrogated and the future has not yet begun, an instant of pure potentiality where everything, as it were trembles in the balance."[54] For Turner, liminality refers to a time-space of transition and transformation, a "between times" experienced at a temporal threshold "betwixt and between" that is simultaneously "no longer and not yet" where it is possible for those inhabiting this time-space to momentarily free themselves from structural time and to access a state of pure potentiality.[55]

For feminist philosopher Catherine Clément, this radical out-of-timeness, or emphasis on discontinuity, relates to the rapturous experience of *syncope* (the omission of sounds or letters from within a word), when "suddenly, time falters."[56] She further explains that although physical time never stops, "syncope seems to accomplish a miraculous suspension. Dance, music, and poetry traffic in time, manipulate it, and even the body manages to do that by an extraordinary short circuit."[57] Clément further reflects on the model of syncope in music described by Jean-Jacques Rousseau: "Syncope: prolongation on the strong [beat] of a sound begun of the weak [beat]; wherefore, every syncopated note is in counter time, and every collection of syncopated notes is a movement in counter time."[58] She elaborates on this movement in counter time further, drawing on Bergson's distinction between time and duration: one closed, static, and inalterable and the other open to transformation and rupture.[59]

Significantly, the model of syncopated out-of-timeness described by Bergson and Clément enables the possibility of a critical perspective on *structural time* predicated on the rupture or destabilization of established *structural values*—indeed, practiced through sudden jumps and jolts rather than through a unified continuity. Therefore, we can assert that syncope—literally, *syn* (with) and *kopto* (cut)—is borne of discontinuity; a missing beat or break in rhythm, out of sync with what came before. In these terms, the out-of-timeness within syncopation neither involves an attempt to escape time nor describes the absent-minded forgetting or loss of time; rather, it refers to a critical practice intent on accessing a different quality of temporal experience predicated on both criticality or rupture. While referencing these accounts of renouncing time is not to suggest that live coders are necessarily "liminal personae" engaged in the ritual transformation of temporal experience, there is evidently a quality of temporal indeterminacy or even suspended atemporality within the practice of live coding where the

liminal condition of *no longer and not yet* leans future oriented toward the opening of other potentialities.

Live coding—like many other improvisatory practices—involves experiential grounding in a temporal present while remaining alert and receptive to the opportunities of the temporal future. Indeed, within live coding performance it is possible to identify different models of possible future states—for example, between a model of the future as *foreseen* (a future-possible that is already planned and scripted in advance of its occurrence) and an ever-emergent future that is endlessly seized and inhabited through improvisatory practice (the near and *living* instant of an emerging future).[60] We hope it is clear by now that we consider live coding to be an emergent force in this way through its ability to rupture the continuity of the linear score and the smooth running of the script. So rather than imagining the future as already existing (somehow *prescripted* and simply waiting passively to be executed), live coding points toward an unforeseeable future borne more of disjuncture, discontinuity, and invention.

In this sense, live coding can be conceived as a kairotic practice based on principles of timing and timeliness, of invention and intervention.[61] The ancient Greek term *kairos* does not refer to an abstract measure of time passing (*chronos*) but rather is often taken to mean the "right time," a decisive moment whose fleeting opportunity must be grasped before it passes. Etymologically related to the Greek word *keirein*—"to cut"—kairos can be conceived as a temporal "opening" or critical moment (a "nick" in time). Here, according to rhetoric scholar Eric Charles White, "Instead of viewing the present occasion as continuous with a causally related sequence of events, *kairos* regards the present as unprecedented, as a moment of decision" where "the flow of time is understood as a succession of discontinuous occasions rather than as duration or historical continuity."[62] He asserts that "kairos stands for a radical principle of occasionality," an improvisational capacity that is "contemporary with itself, alert and able to adapt to the present occasion" where the "subject must always be in the act of creating itself anew."[63] For White, "Understood as a principle of invention . . . kairos therefore counsels thought to act always, as it were, on the spur of the moment."[64] Likewise for Stern, kairos refers to "the coming into being of a new state of things, and it happens in a moment of awareness."[65] He turns to the concept in response to the question: "So, what is to be done with the *now* while life is actually being experienced—while the present is still unfolding?"[66] A kairotic practice is thus critically attentive to the live circumstances or *occasionality* of its own production, capable of creating the conditions *for* while simultaneously responding *to* the contingencies within its own emergence. For us, this comes close to an understanding of live coding and the ways performers tweak their programs as they perform: live coding is a kairotic practice in this very sense.

A further conception of these emergent qualities can be found in the work of political philosopher Antonio Negri, who uses the term *kairòs* (the opportune moment) to describe a mode of immanent (and imminent) invention taking place at the limit of being: "*Kairòs* is the modality of time through which being opens itself, attracted by the void at the limit of time, and it thus decides to fill that void."[67] For Negri,

> *Kairòs* is an exemplary temporal point, because Being is opening up in time; and at each instant that it opens up it must be invented—it must invent itself. *Kairòs* is just this: the moment when the arrow of Being is shot, the moment of opening, the invention of Being on the edge of time.[68]

The future that kairos ushers in is less the not-yet of the future conceived as a continuation of the present as a radical discontinuity. For artist-theorist Simon O'Sullivan, Negri's kairòs can be pictured "as an oblique line—a 'disjunctive synthesis' to use Deleuze and Guattari's terminology—away from the present (but, not, as it were, to an already determined future)."[69] Within this model, "language is creative and future orientated, an exploratory probe of sorts . . . a leap into the *to-come*."[70]

Certainly, these different modalities of future-oriented imagination can be recognized within the field of live coding: for some, the imperative is toward an increasingly predictive future where decisions are anticipated (in advance) to support efficiency, immediacy, and speed within performance, while for others the futurity that live coding seeks to harness is oriented toward a model closer to kairòs, the practice of leaning into the void of the to-come where the unfolding future of the performance emerges simultaneously with its activation. In the latter situation, the live coder introduces individual elements that they understand perfectly but which on execution interact and interfere with each other in ways and at scales that the coder is unable to precalculate and anticipate. This creates a sense of chance that is unlike rolling dice but more like purposefully breaking one symmetry in order to create a new, unexpected symmetry.[71] Approaching live coding through the prism of kairos foregrounds the centrality of attending to an expanded sense of the present therein, emphasizing the flow of real-time action, the criticality of the present moment, alongside both the risk *and* skillfulness required in engaging with the as yet unknown. Drawing on these different conceptualizations of kairotic occasionality, the potential for invention is clear in the sense that the future cannot be prepared for in advance; rather, it *happens*, meaning both that it comes to pass and that it unfolds not through planning or prediction but rather by *hap* (by accident or chance).

For Grosz, too, a future-oriented approach to time foregrounds the attributes of "chance, randomness, openendedness, and becoming."[72] She continues, "Only if we open ourselves up to a time in which the future plays a structuring role in the value

and effectivity of the past and present can we revel in the indeterminacy, the becoming, of time itself."[73] This is perhaps all the more urgent when we seem to have lost an imaginative sense of the future to come, and it is therefore unsurprising that our ability to conceive of change and transformation has become constrained by contemporary forces that lock us into an inert *presentism*, with no recognized past and no real future either.[74] In one sense it could be tempting to imagine the modality of these future-oriented, transformative temporalities as one of acceleration and velocity and moreover, to conceive of the just-in-time nature of live coding's decision-making processes as one that is necessarily connected to the speed of action urgent in the now of the present as it is seized. Indeed, kairos does describe the radical temporality of the very moment of something new coming into being that is unique to that very moment. However, the performativity of timing and timeliness within kairos relies on the *dual principles* of slowness (even hesitancy) practiced alongside speed, the double maneuver of biding one's time and knowing when to act. Paradoxically, perhaps, the critical opportunity within the opening of kairos (ready to be seized) might only be discerned through a slowing down of habitual flows and rhythms, thereby producing the necessary quality of attention in order to act critically. Moreover, as O'Sullivan states, it is this "'affective-gap,' or 'hesitancy' as Bergson understood it, between stimulus and response, which in itself allows creativity to arise."[75]

In these terms, against the too-easy privileging of real-time performance—and the narrowing of the feedback loop between coding and its execution—one could advocate a critical value for the gaps and lags (moments of inaction, sitting back, listening, waiting, and enforced delays) within live coding performance as necessary reflective intervals within which one bides one's time before deciding when to act.[76] In this sense, slow practices are often conceived as a critique of, in resistance to, or perhaps even as the manifestation of *chronophobic* anxiety resulting from the accelerated temporalities of contemporary life. Making explicit reference to the slow-food movement, *slow coding* thus emphasizes the epistemological advantage: "To know code, is to slow code."[77] Experimenting beyond objective measure in such a way evokes durational performances and other modes that attempt to warp time.[78]

Live coding is an experimental practice operating in and out of time. It involves the bringing into relation of seemingly incompatible temporalities: timeliness alongside untimeliness; a sense of temporal continuum (akin to Bergson's durée) alongside the radical discontinuity of kairos; on-the-fly acceleration alongside the biding of time and *organic* time (the subjective sense of lived time) with *inorganic* time (nonhuman clock time, algorithmic *machine time*). Alongside its revelation of the writing of code, live coding reveals and activates the potential of the polyrhythmic, polytemporal, partial,

and uneven microtemporalities operating between, beneath, and below the more readable temporal dynamics of contemporary life. Moreover, it is a practice that harnesses the temporalizing capacity of computer technology as the means through which to conduct its experiments.

The Problem of Ending

Central to this ability to operate both in and out of time—and what we refer to as time criticality—is the challenge to the dominance of deterministic clock time as the means through which time and temporality are commonly understood, especially in technical disciplines and in anthropocentrist thinking. In this final section of the chapter, we explore these ideas by paying more attention to the machine and the ways in which the temporalities of computation challenge many of our precepts of how we consider time to be structured, and even constituted, in technical systems. It is already clear that machine time operates at a different register—for instance, in the way that system time indicates the computer system's notion of the passing of time. In "The Computer as Time-Critical Medium,"[79] Wolfgang Ernst clarifies the ontological importance of time to the computer's operations and networks, from the scheduler to regulate time for computations to the network protocol for coordinating packet switching. He points to key issues of programmability, feedback, and recursion at the level of programming languages as well as the temporal gap between the computer and programmer that is crucial for an understanding of live coding.[80] Precise technical detail is important for the argument—as is, for instance, the flip-flop time of binary switching and the specifics of the clock signal to emphasize that "time counts," as he puts it. For the practice of live coding, timing is clearly paramount and expressed through real-time operations and recursions. Ernst's concept of microtemporality is useful as well to draw attention to time in terms of signals, counting, and calculation.[81] These are some of the complex layers of temporality at play.

If we start from the broad premise that the humanities have tended to overlook technical details, we might draw attention to the way that discussions around live coding have tended to privilege the agency of the human programmer-performer and that other operative nonhuman agents are not sufficiently considered as part of the way in which time is produced and manipulated. Sound synthesis makes a good example, and Ernst refers to the SuperCollider programming environment in particular by quoting Julian Rohrhuber asking: Does "an algorithm for sound synthesis refer to a sonic event or to the machine that created it?"[82] Our larger claim here is that technologies are not simply tools that shape meaning but rather are constitutive of meaning (rather like

philosopher Bernard Stiegler's point that our relation to time is constituted, or "mediated," by the technical means through which it is apprehended[83]). Put simply, we want to insist that the technologies are operative too: that the technical apparatus is both performed and performs as part of any live coding performance.[84] This is what Ernst neatly refers to as "epistemological reverse engineering" to account for the active contribution of machines to knowledge production and how an archaeology of knowledge thereby extends beyond the human sensory apparatus,[85] and what is perceptible, to informatics and the nondiscursive realm of computer programs and technical infrastructures.[86] Although it is the time-critical aspect of Ernst's work that concerns us here, the epistemological implications are significant and something chapter 7 develops in more detail.

Live coding, and in particular its real-time operations, the use of iteration and recursion, offers alternative epistemological perspectives and imaginaries. In his conference paper ". . . Else Loop Forever," Ernst develops this discussion in relation to untimeliness in a rather different way to that which we have discussed so far.[87] His starting point is not the philosophy of time but a fundamental technical concept: namely, the infamous *halting problem* that underpins Turing computation and refers to the problem of whether a computer program, given all possible inputs, will finish running or continue to run forever. In the 1936/1937 essay "On Computable Numbers, with an Application to the Entscheidundsproblem," it was Alan Turing's assertion that a general algorithm to solve the halting problem was not possible. This led to the mathematical definition of a computer and a program, which became known as a Turing machine.[88] This "problem of decision," or "ending," as Ernst puts it, underscores broader notions of algorithmic time and the way the computer forever anticipates its own sense of never ending in an endless loop. That there can be "no happy ending," as Ernst suggests, allows him to elaborate on new temporal structures that no longer align with traditional narrative structures or the terminal logic of the "end of history."[89] In keeping with our earlier discussion, he asserts that kairos is privileged over chronos, and as we have argued, the practice of live coding would seem to emphasize the opportune moment, the point at which invention takes place, which is quite different from the linear steps a computer follows when it executes an instruction in machine time.

Also referring to the decision or halting problem to highlight the unknowability that is inherent in algorithmic music despite, and indeed due, to its deterministic means, Julian Rohrhuber portrays the live coder as working through "multiple alternative trajectories of causality" as a "search for rationality with the means of rationality."[90] As discussed earlier, Rohrhuber points to two ways in which a process can respond to code edits: by responding to state (continuing where the previous version left off) or cause

(recalculating history). Here he notes that these points of decision within loops or recursions also tend to be the points at which timing mechanisms are placed, as with Sorensen's temporal recursions discussed earlier; whether the process loops or recurses is then a time-critical relation of whether the clock drives the reckoner or the reckoner drives the clock.

Temporal complexity is further developed by referring back to Turing's speculation on artificial intelligence and whether a finite-state machine can be aware of its conscious state at a given time and whether a sense of ending is necessary in order to be functional. It is clear that finite-state machines are procedural, in that they produce linear sequences of discrete events in time like a complex clockwork, but as Ernst reminds us: "There is no automatic procedure which can decide for any program, if it contains an endless loop or not."[91] Contrary to the traditional musical performance—with a beginning, middle, and end—Ernst points out that the computational musical recording can be replayed endlessly. Live coding performances also occupy an ambiguous space in this relation because they seem to oscillate between states and thus seem to be most suitable for time-critical analysis. Indeed, the manipulation of time/s is key to this—live coding as the performance of the medium of time itself—where discrete events usually ordered into a sequence for a defined duration are instead open to nonlinearity and entropy. Making reference to Heidegger's "being-in-time" and the knowledge of the inevitable end of life that inscribes a temporal sense of what it means to be a human being, Ernst says: "Humans live with the implicit awareness that their death is already future in the past."[92] This deferral of ending is ontologically exacerbated with computation, unfolding the ending of being as a time-critical condition for both humans and machines alike.

The importance of these ideas for understanding the time criticality in live coding relates to how live coding can be said to activate the present in all its complexity. As indicated by our adoption of the term *time criticality*, we can acknowledge that time plays a crucial role in live coding in terms of the unfolding of time in its performance—as in a timeline or score—as well as in demonstrating how time can be manipulated, and indeed produced, programmatically. If we are to consider programming, the development of Smalltalk, an agile, reflective, object-oriented programming language started in 1971, makes one obvious further reference.[93] What is distinctive about Smalltalk is how it allows for programming with dynamic qualities or, in more technical terms, for interaction with a running system that is not stopped while waiting for new program statements. We look a little closer at Smalltalk in chapter 7, but for now the example helps to establish how a processual practice operates its own particular kind of liveness and temporality that is unique to its technical form and

presents ways to conceptualize how software exists not only in lived time but as actually constitutive of it. This also allows us to shift our attention to both human time and machine time—or rather opens up a tension between cultural and technical registers or that of signs and signals—and, for the critical technical practice of live coding, opens up the tension between musical content and the poetics of the temporal processes in operation. This performative potential of code, beyond its formal logic, is what media/design scholar and experimental media designer Shintaro Miyazaki has drawn attention to in his wordplay *algorhythmics*, referring, on the one hand, to a finite sequence of instructions (algorithm) as a procedure for solving a problem and,[94] on the other, a temporal ordering of infinite movement of matter, bodies, and signals (rhythm).[95] It is not simply the programmer who performs in a live coding performance but a whole suite of technical processes that involve the intricacies of calculation, storage, transmission, and processing in time. These algorhythmic aspects are something that the concept of algoraves further invokes with its reference to the inherent temporal and rhythmic structures of computation and the affective dimensions of bodies moving in space. These algorhythmic interventions draw attention to the complex materialities and microtemporal registers at work in our soundscapes and remind us that machines are running particular sequences and processes that are orchestrated in time.

The concept of time criticality is important, we think, as it articulates some of the ways we experience a multiplicity of presents and presences made operative through the live coding performance. The understanding of criticality we introduced at the beginning of the chapter plays no small part in our argument by drawing attention to the present and to the possibility of actualizing some of its potential.[96] Looping back to the beginning of the chapter, we might go as far as to say that live coding allows for a better understanding of the coming together of different but equally present temporalities and thereby how computation plays a critical role in our ordering and experience of the world—not only how we perceive it but also how it is open to transformation. The attention to computation is important to establish more clearly that time is out of joint, like the crackle of the phonograph that unsettles the distinction between surface and depth and reminds us of the means by which the manipulation of time was made possible in the first place.[97] Live coding points to these more-than-human entanglements and perhaps begins to draw together human and machine registers of time in ways that are not reductive to either.

This is a section of doi:10.7551/mitpress/13770.001.0001

Live Coding

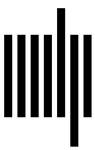
A User's Manual

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

The open access edition of this book was made possible by generous funding and support from the author



The MIT Press

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