humanities

*Article*

# Programming's Turn: Computation and Poetics

**Andrew Klobucar**

Department of Humanities, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA; andrew.klobucar@njit.edu

Academic Editor: Viorel Guliciuc
Received: 17 October 2016; Accepted: 26 April 2017; Published: 5 May 2017

**Abstract:** Digital media and culture scholars routinely distinguish code from any common cultural understanding of media in order to underscore its wholly unique function as an epistemological tool. Where media emphasizes a hermeneutical relationship to knowledge as a mode of interpretation based on its graphic or symbolic representation, the idea of code in many ways invokes a far more complex and dynamic sense of how we determine meaning using symbols or signs in language in terms of producing actual programmable events. In the digital universe, computation, in terms of pre-coded rules, patterns and procedures, continues to showcase all objects and events, along with various corresponding behaviours or viabilities. This paper looks first at a range of contemporary philosophers, like Don Ihde, Katherine Hayles, David Berry and Bruno Latour, in order to build a theoretical foundation for understanding some of the changes in epistemology brought by digital technology and computational reason. Philosophies of computation, I argue, inevitably strive to outline a post-human culture and way of thinking about the world. Although the theoretical weaving of coding with human life follows in part from many earlier modern philosophical discussions on the role language plays in our thinking and sense of selfhood, we can see in computation a very specific reconceptualization of reasoning itself, producing, in turn, a host of new intellectual conflicts concerning human agency and our cognitive faculties. The paper then moves to explore two cultural examples of these conflicts, looking first at the practice of "live coding," a unique, performative event where programmers demonstrate coding before a live audience. Whether on a physical stage in front of an actual audience or simply on screen as a live telecast, such performances combine with coding the distinct habits of gesture and voice in an improvised narrative. One single such show by live coder Sean Colombo is presented here in an exemplary reading of this relatively new media genre. A second, equally significant exploration of similar social and cultural conflicts associated with computation's expansion into everyday living can be seen in the work of the digital literary artist, Ian Hatcher. Ian Hatcher's consistently disturbing video enhanced performances evoke both the structure and overall ambience of a live coding event where he enacts the role of the coder/performer in a process of perpetual conflict with the text appearing on screen. While for many, the live coder can be heralded as a kind of exemplary humanist figure in computation, as these performances show, the more material, writerly aspects of coding must inevitably succumb to the cultural logic of the code's literal execution to produce a distinctly post-humanist approach to writing and art.

**Keywords:** live coding; Phenomenology; computational reason; Digital Literature; poetics; programming

---

In his book *The Philosophy of Software* (2014), David M. Berry takes an important and highly original look at, as he puts it, the "idea of code," describing it in terms of a "super medium" (Berry 2011, p. 13). As such, his project seeks immediately to distinguish code from any common cultural understanding of media (at least within the post-print era) in order to review what might be called its wholly unique function as an epistemological tool. Where media emphasizes a more or less hermeneutical relationship to knowledge as a mode of interpretation based on its graphic or symbolic representation, the idea of code in many ways invokes a far more complex and dynamic sense of how we determine meaning using symbols or signs in language in terms of producing actual programmable events. To distinguish the two forms further, to mediate one's environment, regardless of the format or genre being used, is at base to invent or construct a specific symbolic understanding of it; whereas coding quite literally builds this environment anew.

This apparent division between how text functions in terms of media and as code, which is to say, as a programming tool, infers two equally divergent understandings of text as a conduit for human cognition. To use language hermeneutically in order to mediate and thus interpret the world well underscores textuality's deeply implied cognitive functions. Yet, as software and code become increasingly common as new models for assembling and unifying multiple sources of knowledge, more traditional concepts of writing as self-reflective, critical and canon-based learning practices become less important. Coding emphasizes the use of language, not as a medium for interpreting the world, but rather as a tool for generating it anew through the execution of machine-readable commands. If print-based reading practices helped us imagine the thinking self in terms of eyes and hands proactively engaging with texts on pages, digital reading firmly re-casts human cognition as eyes interlocked to screens of all sizes, attempting to access moment to moment the endless stream of transmissions our codes are executing. This distinction, as this paper will show, provokes a host of new aesthetic challenges within the literary arts. If writing practices in general depend upon the text as media, offering writers—through their respective eyes, hands and whatever writing utensils may enhance these senses—the opportunity to question, interrogate, build and rebuild phenomena, an intriguing conflict quickly emerges in relation to code. Code, being neither phenomena nor media, is arguably not open to interpretation in this way. Hence, as we'll see in the work of specific digital poets like Ian Hatcher, aesthetic engagements with actual programming automatically introduces new textual elements counter to any media forms, openly positing computational procedures as inherently dissimilar, if not opposed to human cognition.

Berry sees this opposition as significant to understanding both concepts—i.e., media and code—to the extent that not only is it inaccurate to qualify one as a type or even instance of the other, but that, as sign systems, they might actually function as antinomies of each other. Coding, considered here in Berry's critique, easily recalls something akin to a distinct mode of being. In fact, to refer to the act of coding in terms of how it operates necessarily describes, for Berry, a highly developed "computational ontology." In his words, to understand code demands that we place it "firmly within the larger political and social issues that are raised in contemporary society...[and the] profound ways in which computational devices are changing the way in which we run our politics, societies, economies, the media and even everyday life" (Berry 2011, p. 10). All such traits contribute to Berry's crucial qualification of code as a sign system able to operate beyond the typical attributes of media—that is, as a "super medium." Media continue, of course, to interact with code, providing a plethora of individual modes of interpretation. In fact, the very possibility of inputting information ever more productively in our current cultural moment requires the use of media in an increasing variety of formats. Despite coding's supposed autonomy from the material world around us, we must continue to improve an ongoing "communicative dimension" with our various handheld and wearable devices in order to compute our lives better and more inclusively in the service of knowledge.

Contemporary philosophers and cultural theorists alike have long recognised media's essential role as a hermeneutical tool in modern epistemology. Mediation, as every discipline in science since Galileo pointed his telescope upwards rather than seawards demonstrates, is crucial to understanding each "thing" we decide makes or takes up a place in our world. In representing the world, however we choose to do so, we are, in fact, establishing it, making it meaningful—calling it, if you will, into existence. But this relationship, having derived exclusively from a non-computational world, Berry makes clear, remains newly challenged by the use of code in terms of how we relate knowledge to sign systems. To compute the world is also, of course, to bring it into existence, yet such an exercise differs from any "calling" or signaling gesture, where real objects or events somehow acquire their visibility—become discernible—first as media. The physical act of experiencing heat in the air around us may lead us to suspect it is, in fact, a warm day, but only a thermometric reading of the air will confirm this warmth as real or actual. In this paradigm, mediation remains necessary to interpret the world beyond felt phenomena. Computation, by contrast, effectively flips this relationship upside down, presenting the signal quite literally as a mode of generation. Data inputted as code for that same warm day effectively produces the very same heat we may subsequently (or not) feel; in Berry's words, computation occurs when "a model or image is stabilised and attended to, and then internally transformed depending on a number of interventions, process or filters and then outputted as a final calculation" (Berry 2011, p. 10). Whatever terms and examples we may use to describe computation as an epistemological practice, they will inevitably share a fundamental distinction from representation, emphasizing instead a configuration of the real-world as an unfolding, generated situation that is neither a fixed, transcendent, a priori structure, nor its media representation. Coding presents instead a somewhat anomalous third type of activity, and with that activity, a new mode of cognitive engagement, complete with its own nature of understanding.

It is precisely this revisionary appeal to cognition that inspires the theorist N. Katherine Hayles to declare our prior epistemological regime based upon objectivity and its individual, sensual interpretation to have formally concluded, giving way to what she calls the "computational regime." Hayles uses the term richly throughout her two best known books, *My Mother was a Computer* (2005) and the more recent *How We Think* (2012). Looking closely at the profound impact of code on everyday life, *My Mother Was a Computer* crucially compares its social function to what we previously attributed to speech and writing, while emphasizing their profoundly different capacities as both sign systems and epistemological tools. By entangling the communicative and epistemological roles we once held as exclusive to language with coding (and thus computation), Hayles maintains, we have effectively transformed the very nature of reality itself, at least in terms of what it means to us. Although coding, whether we are writing it or even simply running it as software, can seem to be performing as a kind of language, it involves wholly novel interactions that cannot be qualified as strictly linguistic. In fact, when considered as a kind of material event in itself, coding might accurately be related to entirely new modes of perception, neither wholly symbolic, nor physical in structure. Accordingly, for Hayles, these relations operate at conscious and unconscious levels, involving interactions she describes as "dynamic and continuous, with feedback and feedforward loops connecting different levels with each other and cross-connecting machine processes with human responses" (Hayles 2012, p. 13). Computation invites us to consider nothing less than a new material existence—a new *digital* reality, where, Hayles further points out, "the object itself is not a pre-given entity but rather a dynamic process that changes as the focus of attention shifts" (Hayles 2012, p. 14). Gone with our previous universe of "pre-given" objects that tended to prompt both a physical and mechanical mode of interaction is any corresponding sense of meaning as a time-based, cognitively identified proceduralism. The digital universe offers instead a very different form of reason—one based entirely on algorithmic patterns and configurations of any number of discrete elements or variables.

To better understand this mode of reasoning, we might look to the theorist, Albert Borgmann (Borgmann 1999), whose foundational study *Holding on to reality: The nature of information at the turn of the millennium*, 1999 aligns the world as computation with whatever underlying codes apparently running it. This concept of an essential programmability behind all reality stands in marked contrast to western philosophy's traditional references to a universal "Logos." Our ultimate epistemological aims, whether we define them in terms of a grounding principle of truth or socially vetted objectivities, in the pre-digital information economy inevitably privileged what Jacques Derrida describes as the "transcendental signified." Key to our analysis here is the central role this principle has historically played in establishing a basis for writing itself as central to knowledge's ongoing determination. While in this paradigm we may not be able to determine any ultimate "signified" sponsoring our experiential world, we can nevertheless continue to mediate it through our representations, our writings or tracings—whatever forms we imagine capable of bringing forth some kind of relationship (political, spiritual, social, etc.) to the Real beyond. Programming functions according to a startlingly different logic. As Hayles reminds us, computation bears in essence a very minimal premise, holding at base nothing more complex than an "elementary distinction between something and nothing (one and zero) and a small set of logical operations" (Hayles 2005, p. 23). In this binary paradigm, all relationships to knowledge as a mode of signification, in other words, as a consistent and ongoing epistemological shift between what is written and what isn't must be similarly abandoned. To define being in essence as an either/or state—again, to quote Hayles, as either something or nothing— importantly repudiates our long established epistemological association of objectivity with signification, calling into question the concept of mediation itself as a means for determining the very extent and "realness" of our actual world. The ones and zeros of digital computation work fundamentally differently than what we typically understand as modes of symbolic representation in that the patterns they produce via alternating signals of "off" and "on", something or nothing, cannot be considered to signify or somehow render prior entities. In fact, it might be more accurate to interpret them literally as entities in themselves. No preceding state, save the essential code or program determining emergent patterns can be separately identified. And again to look critically at this very code along with any patterned activity it subsequently generates necessarily requires we differentiate both types of structure from anything figural. In replacing the transcendental signified, and with that figure any a priori philosophical focus on a universal "Real" or sense of "Being," with code, computation not only supplants western culture's institution of the Logos, it questions our entire social, intellectual and even political relationship to writing in general. Writing considered as a tool of mediation and knowledge construction invokes a very different relationship to coding compared to the image of the transcendental signified. Founded on programming sequences rather than a metaphysical sense of symbolic or semiotic coherence, the entities we find in our new digital universe cannot but appear comparatively elastic and changeable. Coding, as both a practice and also its own platform for working with variable data, is not designed to produce distinct, individual materialities in the world. Rather, the very structure of algorithms, which is to say the essential "computability" of data, evokes at base a more or less constant process of execution. The algorithm, in other words, is synonymous with operation, not definition, its primary order being the system of rules that direct its various functions. To be defined by computation is to be in constant, ongoing rule-based calculation; outside such procedures we can imagine only the possibility of some later configuration. Such a state, too, contrasts with more traditional understandings of materialities and forces beyond the signified as essentially chaotic and disordered. To be outside computation is not to be equated with a state of chaos, i.e., a lack of (and therefore call for) order. In the digital universe, where computation becomes a literal condition of being, the non-computational refers accordingly to that which isn't. Rules, orderings, procedures continue to exemplify states of existence, while the entities or events subsequently produced by such codes showcase a variety of corresponding existential behaviours or viabilities. In this paradigm, even our traditional sense of organic lifeforms as distinct things in themselves, with each one exhibiting a certain degree of agency or autonomous will seems questionable in favour of newer concepts of life as a kind of constantly shifting and evolving bio-physicality.

Some of the more progressive philosophers of this new sense of environment, like Don Ihde, Evan Selinger and Peter Paul Verbeek—see, for example, Ihde and Selinger (2003) and Verbeek (2013)—continue to emphasize in their respective works a more stream-like model of life, calibrated not in terms of distinct entities, so much as according to moments of behaviour. For Verbeek, this is especially significant when we consider how profoundly new "visibilities," regardless of the medium from which they are derived, can affect any subsequent sensual understandings of the world and its myriad processes. The fact that "technologies fundamentally shape people's experience of disease, pregnancy, or their unborn child," Verbeek argues, should not be misconstrued as examples of media providing us with more insight into the physical world itself (Verbeek 2013, p. 9). Rather, these revised visibilities might best be interpreted as an entirely new range of understandings and concepts as much in dialogue with technical constructions as with bio-sensual events. Verbeek phrases it particularly well when discussing the obstetric ultrasound: "this technology is not simply a functional means to make visible an unborn child in the womb. It actively helps to shape the way the unborn child is humanly experienced, and in doing so it informs the choices his or her expecting parents make. Because of its ability or make visible the fetus in terms of medical norms, for instance, it constitutes the fetus as a possible patient and, in some cases, it's parents as makers of decisions about the life of their unborn child" (Verbeek 2013, p. 16). We have what Verbeek calls a "fundamental intertwinement of humans and technologies; in his words, "human-world relationships should not be seen as relations between pre-existing subjects who perceive an act upon a pre-existing objects but rather as sites where both the objectivity of the world and the subjectivity of those are experiencing it and existing in it are constituted" (Verbeek 2013, p. 15).

Ihde's concept of bio-physical intertwinement compares well to the relationship that Hayles, along with other contemporary philosophers like Gilbert Simondon, Bruno Latour, ascribe to living in code—perhaps even as code. Again, in many ways, this theoretical weaving of coding with human life follows directly from, while addressing what Derrida termed the transcendental signified as a linguistic source of being and knowing in modern philosophy. Coding, by bearing in and of itself a sense of intrinsic root structure or ordering principle, actively counters the ontological role of language we see in Derridean thinking. To work with coding, accordingly, infers none of western philosophy's consistent emphases over the last century of language as an epistemological tool in support of an active hermeneutics. Heidegger's remarkable allocation of language as "the house of being" (Heidegger 1998) remains probably the most succinct and plainest distinction of its essential relationship to the Real. In relation to code, however, language emerges newly associated with the non-computational, referring for the most part to any and all symbolic modes communication that have no active role in determining reality. Even when looking at various source codes in a program, or the written record of a functioning programming language, the key lines of any working script cannot be considered "readable" in a typical manner. Most messages derived from programs remain secondary to the overall function of the code itself. They may provide comments about the lines of code running, or simply instructions and "read me.txts" designed to assist in the program's installation and ongoing execution.

Code thus presents a consistent challenge to traditional media theory and philosophy, especially in terms of media's role in epistemology and the construction of knowledge. As many of the theorists mentioned already, including Verbeek and Hayles, directly contend, computation offers a very specific reconceptualization of reasoning itself. It may be tempting, of course, to compare this mode of reasoning as computation with more traditional models of instrumental thinking in western philosophy. Instrumentalism has long supported the epistemological principle that thinking or reasoning as an activity best be regarded as an execution of sorts, in terms of providing a means to gain specific aims or predict phenomena. The philosopher David Golumbia, for example, argued that computation automatically favours an instrumentalist understanding of culture, since it demonstrates formulaic relationships between variables (Golumbia 2009). To compute phenomena is at base to determine specific ends from certain means. What could more instrumental than formally adding ideas together to produce procedurally credible results? Berry, however, refutes the claim that such arrangements constitute forms of computation. Where instrumental rationality, he argues, refers to "the application by an actor of *means* to *ends* through mathematics, empirical knowledge and logic...computational rationality is a special sort of knowing, it is essentially vicarious, taking place within other actors or combinations and networks of actors (which may be human or non-human) and formally algorithmic" (Verbeek 2013, p. 13). Hence, for Berry, computation's relationship to language is much more exclusive compared to the typical usage guidelines we see in instrumentalism. Computation in this model should be considered as functioning more or less autonomously from language, with its execution bearing little significance with respect to any one media format.

As I want to argue here, to work with code, whether in terms of an active programming practice, or simply encountering it in the form of daily logins, encryption requests, permissions for access, or the inevitable software crash, is to invoke a highly distinct relationship to language. In fact, because coding, culturally speaking, provides an enduring and typically unquestioned allusion to qualities like reason and intelligence, it seems especially important to emphasize its symbolic of discrepancy from actual cognition. For the most part, as a mode of locution, coding involves little, if any, formal semantics. There are no references, cultural connotations, much less shared social perspectives or values to be gathered through our collective login activities—although passwords and personal identification data may allude here and there to certain personal associations. Even transactions of this type describe in essence an exchange with digital sequences, temporarily translated into alphanumeric strings of non-interpretable symbols. Thus we can accurately ascribe the "meaning" of code to derive in some ways from a different, non-linguistic paradigm of representation. Inputting the right string in the right order in the right form, as any coder will maintain, epitomizes a distinct mode of data exchange. We may liken this type of interchange to literal communication in the sense that it too often seems as if we are in some form of oral conversation with our digital devices. As gamers will easily confirm, we may find ourselves relying on our usernames more frequently than our proper names when formally identifying ourselves over the course of our day-to-day activities. For certain communities this means of personal reference is already considered the most accurate and trusted one. How many of us think of ourselves first in terms of our social media IDs? Is this not what certain theorists like Berry and Hayles describe as what it means to live with code—or even as code? Our usernames, our sign-ins, our seemingly endless pleas for access to an equally interminable number of mobile networks collectively normalise our day-to-day existence as one constant stream of data flow., And while such an environment is typically described as a choice-laden, exercise in human reasoning, it seems just as important to distinguish these interactions from prior, non-digital modes of communication and perhaps even language use in general.

Addressing this last point, the French philosopher Bernard Stiegler describes digital computation models as an ongoing "systematic discretization" of the world itself, both in terms of its material processes and singular entities (Stiegler 2007, p. 149). In Stiegler's words, paradigm, computational processes epitomise an ongoing "grammaticalization of the visible," which is to say, the construction or "encoding" of any and all phenomena into symbolic sets of "discrete" data (Stiegler 2007, p. 149). Consequently, coding as a transaction invokes, not just a new means for communicating, encouraging us to consider it as an effect or even sub-set of linguistics; rather it may at times even appear to go beyond the ontological scope of language to re-ground our very sense of reality, especially in terms of how we perceive entities interacting with each other. To compute our reality, in other words, is to validate it according to a different kind of grammar. What bears more analysis, via Stiegler's line of thinking, is how this particular grammar, perhaps best described as algorithmic, forgoes certain limits of language as a mediating tool to offer instead a modality capable of generating the very data that constitutes many aspects of reality, regardless of its ever-varying and multitudinous epistemological contexts. Language, considered together with technique and thus technological systems, signifies not just how we perceive reality, but how we build it.

At times, language use, whether written or oral, seems to involve a more complex, perhaps even conflicted relationship, with computation. In fact, in the digital universe, writing typically invoke fundamentally flawed algorithms, while misidentifying many of coding's more essential structures and operations. The most common example of this misidentification may be the tendency among users to construct passwords that are too semantically recognisable and therefore vulnerable to electronic theft. Much safer, users are constantly advised, to choose symbolic variation in format: i.e., mixed upper and lower cases, or more equal ratios between numerals and letters. Indeed, language often appears as something of a deficient symbolic conduit to be suitably enriched in the computational world, a flaw in signification to be corrected through more efficient algorithmic processing. These procedures are becoming more apparent in neuroscience, where we typically think of communication in terms of pre-coded behaviors, expressed algorithmically. Whatever writing's role in a world increasingly reliant on software and the code directing it, computational reason automatically problematizes its relationship to knowledge as inherently inadequate, perhaps even misguided in its implication that rationality is a mediating faculty. To reason as a "computer," that is, as one who computes, at base refutes signification as a means of construction in order to invoke its capacity as a tool of execution. With code operating quite literally as a determining agent of the concrete real, and all relationships functionally "algorithmic," signification seems again best akin to what Stiegler calls a mode of symbolic systematic discretization. Language, by contrast, appears as little more than a flawed mode of access, inviting us in our dual, perhaps even conflicted capacity as both programmers and system developers to critically interrogate all user interface. To these ends, knowledge corresponds less to the exchange of information than the transmission of digital signals. All other non-computational modes of mediation, which is to say these signals operating outside the code, will inevitably seem secondary. Once the foundation of what Heidegger called the house of being, language has effectively collapsed in a new regime of linguistic foreclosure. Here, on these long abandoned semantic estates, media theorists, too, will find themselves freshly re-consigned from their past intellectual responsibilities to the primary task of patrolling the borders and modes of user interface (UI) access.

One only has to look to contemporary developments in UI design to gain important insights into the challenges code presents as a symbolic object of analysis. Despite ongoing technical advances in interface software, UI's relationship with coded behaviors remains characterized by limits and inaccuracies. Some of the clearest indications of the conflicting aims between these two practices are especially evident in the exercise of "live coding," a forum or event where programmers literally demonstrate coding before a live audience. One of the more active communities in this practice, a network known somewhat variably as "Transnational, Terrestrial, Transdimensional, Temporal Organisation for the Promotion, Proliferation, Permanence, Purity, Parsimony, Pragmatics of Live AudioVisual, Art, Artistic Programming" (TOPLAP 2016), collectively celebrates such events as uniquely revealing in their capacity "to expose and rewire the innards of software" (http://toplap.org/about/). Alternatively described as "on-the-fly programming," live coding commonly situates itself within the performing arts, casting the programmer first and foremost as a creative artist capable of demonstrating a personal as well as unique skill in coding.

Throughout computational culture, the live coder heralds an exemplary humanist figure, whose programming efforts, in turn, provide a model for an intellectually balanced relationship with the digital world. Whether on a physical stage in front of an actual audience or simply performing live on screen, such characters bring to the field distinct habits of personal gesture and voice, as they colorfully navigate their codework in improvised narratives. In many cases, live coding is performed in a reluctant, half-audible mumble under the coder's breath, countering images of cold technology with themes of human fallibility and uncertainty. Just as often, the narrative takes on the tone and energy of a live sports broadcast with play by play recount. Both performance styles show us how even a configuration as mathematically and architecturally unforgiving as software can inspire—perhaps productively incorporate—gestures of interpretation, speculation and error. In short, in live coding, computation appears first to re-acquire many of the qualities and characteristics of language theorists like Berry and Stiegler previously have distinguished it against. The live coder, sampled at just about any random point within these performances, seems perpetually caught in mid-conversation with the very program taking shape before us. Questions concerning terminology, punctuation and, most vigorously, the very logic of statement ordering, especially with respect to recursive tasks, flow freely as each line materializes. A voice, human, all too human, emerges above the calculating, incessant din of ones and zeroes flowing from task to task, and with that voice, the tell-tale signs of personality, skill and, at times, perhaps even artistry. Petitioning for some level of re-admittance into Heidegger's damaged house, coders quite naturally begin their offers with several well placed references to aesthetic theory, evoking the value of individual expression, innovation and, of course, taste. Geoff Cox, who has been theorizing coding in relation to speech and language for close to a decade ascribes to programming a particular "aesthetic value" based on "the element of time [by] combining improvisation with algorithmic configuration" (Cox et al. 2004, p. 163). If the code itself is subject to improvement or at least some recognisable transition in shape or form towards a distinct end, then, Cox implies, it is on some level distinct from the software program it appears to be executing. It acquires, interestingly enough, a history of its own; and history, being "inherently unstable, contradictory, dynamic and dialectical" remains first and foremost a tool of interpretation. At this point, Cox's analysis shows us, code is decidedly more linguistic than mathematical.

On one level, Cox's argument follows an important historical trajectory in the cultural analysis of coding emphasizing linguistic materiality in all programming practices. Donald Knuth compared coding to "composing poetry or music" almost fifty years ago given its aesthetic appreciation of grammatical form and the "economic" placement of line (*The Art of Computer Programming: Volume I, Fundamental Algorithms*, (Knuth 1968, p. v)). More currently, organisations like The Alliance of Code Excellence (ACE) continue to argue for correctly and well-written code able to demonstrate a formal and thus highly "readable," material coherence in its presentation. For this organisation, as with writers like Knuth, the materiality of code as language also endows it with specific standards as you would expect with any media system and corresponding practice. To write code is at a fundamental level to engage a pre-set system, more or less successfully, in relation to certain underlying, predetermined and thus assessable criteria—in this case, the best possible execution of a software program. And yet as I want to argue here, emphasizing these more material, writerly aspects of coding must inevitably evoke a very distinct contradiction in the practice in the sense that, unlike with other media forms, we have at our disposal simultaneously a predetermined set of events, as conveyed by the program's literal execution, pitted against an evolving gesture of communication, as conveyed through the code being written.

The resulting paradox between the unpredictability of active communication and computational precision remains essential to any live code performance. To demonstrate this paradox, we can turn to one of the most popular resources for such performances: the webcast platform "Livecode.tv". Tune in to a live coding broadcast on "Livecode.tv" and you are treated to a "front and center" viewing of individual programmers writing code live at a terminal for their own individual software projects, most of them related in some way to gaming. Audience sizes range in number from a single dozen or so watching an interface designer like Nikita Kiselev build a Drupal site for a manufacturing plant in Russia to tens of thousands of onlookers actively following a gamer like Sean Colombo develop a new fantasy role-playing board game he titles "Tatsu" to be distributed by Valve on the steam platform.

For several months in early 2016, Colombo maintained a surprisingly consistent audience of gaming enthusiasts, while he refined specific elements of his project for all to see and comment on in real-time. Consistent with the fantasy genre, Tatsu features a traditional medieval setting, derived, as the name implies, from Japanese folklore, complete with feuding kingdoms and mythological references to dragons, temples and certain primal elements like fire and water, etc. Colombo's work, once available and released to market, will offer players the opportunity to sit online around a single, virtual board game before taking on the identities and behaviours of various dragon characters that they, themselves, have managed to construct for this shared environment. During one particularly robust session recorded in May 2016, Colombo appeared particularly focused on making sure the game board would be easily viewable over the latest Apple operating system "X El Capitan," released that same month. Clicking on Colombo's channel (https://www.livecoding.tv/seancolombo/) (Colombo 2016) at just about any point during this session finds him well situated according to its typical network format—which is to say squarely positioned as a headshot at the bottom right side of the screen while lines of code flit consistently left to right across his terminal for all to see. Viewers are simultaneously invited to "live chat" as the performance continues. For the most part, however, this audience seems particularly rapt, as few comments are forthcoming save once every 20 s or so. And even among these offerings, a surprisingly small number seem related to the project at hand. One onlooker identified on screen as "Locode" joins the room 16 minutes into the session and promptly informs Colombo that his code "looks like a big black hole." Another person simply introduces himself as a viewer from Uzbekistan before challenging Colombo to prove he knows where the country is located. Colombo makes an attempt, but fails much to the Uzbekistanian's delight: "It looks so fun, when a man from

USA is trying find information about Uzbekistan," he observes. A wry comment on the current deterioration of diplomatic relations between the United States and the Russian political sphere? Whatever the case, Colombo declines to push the conversation forward and continues working on his game in the Visual C+ and C# programming languages.

Looking at the actual program, we see that viewers are treated to an image of Colombo capably tapping out line after line of C# code in the centre part of the screen. There are at least three archived sessions of his channel on the site itself, each of which features a similar dynamic. Colombo codes actively, demonstrating in the process his own planning and tactical vision as he builds this project further. A more focused review of a specific two hour session, recorded one afternoon in January 2016 as "Steam/C#gamedev:internationalizingmygame," gives us insight into Colombo's typical level of engagement and in many ways exemplifies the type of programmer the critic Erkki Huhtamo calls a "software art purist," which is to say nothing less than a creative writer for whom the primacy of code is considered the main artistic material. When Colombo works through various edits in his code, clarifying terms, and building new "scripts" to take care of multiple repetitive tasks, he is essentially improvising for a specific audience simultaneously within the confines of two different, but equally programmatic, as well as shared language systems.

As Cox notes, "any genre of improvisation relies on a predictive understanding of complex and generative systems," and Colombo's performance of live coding makes this relationship especially clear. His objective for just about every task he habitually checks off his regularly accessible "to-do list" is first an exercise in grammar and syntax. While he may comment briefly on specific behaviours or interface effects being sought, the "live act" itself remains expressly syntactic. Different terms, being coded commands, are checked and re-checked for consistency. If the programming syntax for a particular language like Visual C+ or C# is inconsistent, producing rendering problems or ruptures in game sequences, Colombo's improvisation centers on how quickly he is able to locate and fix them all. 20 min into the session, Colombo observes for us

> Um, alright; so I'm still trying to figure out how we actually rendered 'that' over there. So there's all these lobby slots and somewhere they have to be changed into views, sprites and things. One way to find it is to go look for the strings (pausing while looking over his screen) "lobbyslot view"... interesting. So, I'm going to look at these "lobbyslotviews" and I'm going to see where they get their size... (18 m 50 s–19 m 02 s)

Fellow coders watching him as he searches for the right term will typically understand the particulars of each task being performed. In other words, they know that when playing over a network, gamers must occupy distinct "slots" in what is usually termed a "multiplayer lobby system." The primary grammar in play here holds the word "Slots" as one particular variable associated with its key term "NetworkLobbyPlayer," a label designating any individual player in a single lobby system. Again, the relationship Colombo is describing when analyzing his work in this instance is primarily syntactic, and any narrative tension that subsequently emerges seems comparable to that associated with any writer in the process of refining the grammar of a text. If his audience hopes to learn anything from this particular recording, it will be similarly useful grammars and syntaxes to produce their own respective games in the C# language.

By "coding" Tatsu, Colombo is at one level authoring a narrative much like any other writer in the sense that his game will invite viewers to engage with a world of his own invention, complete with many different scenarios, and numerous fictional characters and plot lines. C# remains his medium of choice for completing this project, and Colombo wants to demonstrate with his performance both his proficiency in the language, and also how effective it is to produce the type of gaming experience he values. At another level, however, Colombo's coding demonstrates an alternative, non-semantic use of language in the sense that Tatsu, describes a rich display of computational behaviours and reasoning. The work being generated within the game can, in other words, be understood literally as a set of calculations. To witness Colombo in action as a coder is to invoke an alternative mode of production based upon the ongoing computation of data through specific algorithmic formulae. The fact that a video game is being coded automatically underscores its capacity to generate structure after structure according to very precise digital systems of its own. Such calculations remain by definition autonomous from either Colombo's or any subsequent player's interactions, and, as such, are rarely invoked, whether through the chat box or his ongoing monologue. The game, being ultimately a mode of computation, functions as something other than communication, whether written or oral. To compute, as Hayles and Berry maintain, is to operate outside the very limits of representation. Once a program is run, once routines are executed, any number of freshly generated patterns of activities and events may be set in motion distinct from the different languages (like C#) we need to communicate with this program. And while these very activities and events will continue to appear on our terminals in many different media formats, including audio, graphics and even text, the computability sponsoring them remains a separate, inherently hidden activity. Live coding, consistent with Colombo's performance, quite effectively evokes an essential capacity in all computational programs to operate our projects at hand, dramatizing the power of code to shape and even create our cultural environment with a unique and indisputable precision. At the same time, whatever modes of representation, whatever tracings as written languages, whatever user interfaces or UIs, we employ within these environments, we bear a fundamentally incompatible relationship to programming itself as a generating tool. The events we see before us, the media we use within this digital landscape continues to indicate that something else is happening. Non-visible, non-representable transactions, calculations, formulations are occurring, evoking an interesting conflict in every line the live coder offers for us to interpret.

Presented in this way, computation itself re-emerges as a far more complex and dynamic structure. The coder assumes a uniquely prominent place, resuming his or her place as a creative author. But this paradigm, although enabling, tends to overlook the distinct, invisible, non-linguistic role of algorithmic execution in the process. Beyond our material efforts as writers to communicate is precision, not void or chaos, as modern philosophy has long held. The sleep of reason is more reason. Hence coders inevitably face in their respective practices a very new environment of conflict. The gap in place here is not necessarily aesthetic in the sense that no cultural link between the computational and non-computational is forthcoming. Despite how flawed and frustratingly inconsistent these attempts to correct and refine the codes behind the computation are, the computational event itself is always by definition pre-determined. In this sense, the code is at some level always exact. Even a non-functioning program is at base programmatic.

Some of the more advanced code-based, electronic literary projects seem to capture this interesting paradox in computational culture particularly well. Ian Hatcher's consistently disturbing video enhanced performances evoke both the structure and overall ambience of a live coding event with him enacting the role of the coder/performer in a process of perpetual conflict with the text appearing on screen. In one specific piece, "Drone Pilot" (2015), Hatcher engages specifically with the flow of text projected above him. For the first 30 seconds of the reading, his voice closely corresponds with text:

> When you see nothing connecting you say nothing connecting (Hatcher 2016a). (http://ianhatcher.net/#!/video)

The lines (along with the poem's title) immediately call to mind contemporary U.S. urban policing messages in defense against possible terrorist attacks. Before long, however, the visual presentation of the work begins to overwhelm Hatcher's vocal performance, running faster and faster in a cascading, almost torrential flow of increasingly blurred text. Seconds into this shift, Hatcher's own reading pace amazingly enough begins (at least in appearance) to match the animated column behind him. I say "in appearance," as the column can in no manner be read by the audience at this point, much less spoken aloud. As if to confirm the work's essential non-readability, the column itself begins to shift in width, shrinking and enlarging its parameters until lines pare down to mere letters before disappearing altogether. Consistent with the letters vanishing on screen, Hatcher abruptly covers his mouth in dramatic bursts of silence. Thus can code be "voiced," so to speak, in the sense of an actual text animation program being read as accurately as possible.

A similar dynamic can be glimpsed just as plainly in Hatcher's use of the page, as we see in most of the poems collected in his first print publication, *Prosthesis* (Hatcher 2016b). Throughout the collection, Hatcher encourages us to take issue with the human voice and how it is represented—in many cases, misrepresented—in digital culture. In one of the first poems, a piece entitled "Pyramid," for example, introducing the book's "Input" section, the very challenge of locating a distinct subject in the text becomes one of its primary themes. A kind of "self" emerges almost immediately in the first stanza: "distinctions i edges/morning i following/never i this an unbecoming route" (Hatcher 2016b, p. 12). The grammatical structure of the written work, however, defies the logic of speech in favour of a programming syntax. This isn't a Perl poem as conceived by the well-known (and self-described) "Perl poet" Sharon Hopkins, where the text functions simultaneously as a readable work as well as coded instructions in the Perl language [1].

Hopkins's work provides an interesting dual function, setting an executable code in tandem with a list of readable comments; by contrast, Hatcher's poetics emphasizes a conflict between data types. Words like "distinctions," "morning" and "never," function semantically in the poem, calling to mind specific images and feelings in the reader's mind. At the same time, their syntax as code delimits values to which a set variable "i" exhibits a number of behaviours in the following order: "edges," "follows" and "this an unbecoming route." One cannot read meaning in these lines without "processing" them due to their resemblance of a source code. An "i" is called upon to "edge" a "distinction;" that same "i" is also "following" a "morning;" and finally the "i" must "this an unbecoming route" to "never". At the same time, the lines demonstrate a certain aesthetic sense of linguistic arrangement, the pronoun "i" evenly dividing descriptive references to boundaries and traces of time and space, evoking a kind of pentameter in the opening lines.

> distinctions i edges
> morning i following (Hatcher 2016b, p. 12)

Quite literally Hatcher presents us with a computational self, albeit from a non-computational paradigm. One of the primary conflicts in the poem evolves from this specific paradox. A syntax error occurs on two fronts: as a non-computational statement, the conscious subject, the "i", of the poem appears hopelessly fragmented, uncommunicative and confused. As a computational routine, the lines fail to run. Further on in the second stanza we have

> do while
> [i] continue
> clouds of slate-gray
> obscurant sand pouring over
> pyramids timeless buzzing wonders (Hatcher 2016b, p. 12)

The same approach opens the stanza up: "do while" makes sense syntactically as part of a coded subroutine or procedure where a specific "condition" is being programmed. "Do" one task or provide one value while another one (in this case, our paradoxical "[i]") performs something else. Hatcher includes a tool "appendix" at the end of the collection where the phrase "do while" is formally identified as an instruction to "loop through a block indefinitely until the specified condition is false" (Hatcher 2016b, p. 148). Again the syntax, despite referencing literal functions, is in no manner fully computational. However, as with his live performances, it is precisely this uneven quality, in other words, the text's failure to parse specific functions correctly as actual software that conveys much of the work's cultural import, and as described above overall thematic tension. At best, the syntax calls to mind a possible computation, or perhaps the need for computation. If parsed correctly with a better,

[1] One of the more prominent and widely reproduced pieces of this genre remains her 1995 experiment "Listen," (Hopkins 1995) as published in both the *Economist* and the UK national paper *The Guardian* in the same year. The text of the poem is shown in its entirety in an Appendix A to this paper.

more consistent set of statements, "i" might return values within a fully functioning subroutine. But this poem, especially when considered more in terms of a UI, simultaneously breaks the code to call to mind both a listed bullet: i.e., [i], [ii], and a kind of Wordsworthian subject gazing thoughtfully upwards at "clouds of slate-gray." At this level, too, one must no doubt see the poem as highly dysfunctional. To hear Hatcher read it is to witness an almost violent struggle between a single Cartesian self and a set of executable functions. His voice, as with "Drone Pilot," laboriously contradicts traditional speech patterns and tones, not so much this time via erratic pauses and a maddeningly machinic pacing, but rather through his chillingly convincing imitation of artificial speech-audio technology. Poets, sound artists and even musicians have long relied upon text-to-speech software to produce this effect. The coldly de-humanized tone of these audio tools, especially when cast as a male persona, is decidedly convenient as a cultural symbol of computation-based social authority and instrumental reason. To hear Hatcher's performance of this figure remains for the most part equally terrifying and beguiling. In many ways, "live coding" presents the best genre for describing the respective literary and performative values of these pieces. One's sense of the work continues to underscore computational and non-computational structures as cultural spaces intrinsically at odds with each other, while we as readers must nevertheless occupy both in terms of how we define selfhood as well as of the boundaries of all social interaction.

Various important theoretical connections continue to link the live coding work of gamers like Sean Colombo to the poetics of Ian Hatcher. Both performers quite deliberately build an impressive set of aesthetic statements around the inherent conflict between language's accepted function as media and its more current, ever-growing capacity to compute entire worlds of being. The very concept of consuming code as culture, inviting it into our minds and bodies via both page and screen carries with it a host of interesting challenges to what it means to read, interpret and, of course, understand writing. At a fundamental level, as we scan each line of text, we must pause to consider to what extent the writing is, in fact, reading us. Such questions remain essential to all live coding performances, regardless of whether the object being performed is a Hatcher poem, a "steam" game, or any new software project under development. Each work carries the fundamental core message that we are simultaneously both live and code.

**Conflicts of Interest:** The author declares no conflict of interest.

## Appendix A

Sharon Hopkins "Listen" (1995)

```
#!/usr/bin/perl
APPEAL:
listen (please, please);
        open yourself, wide;
            join (you, me),
        connect (us, together),
tell me.
do something if distressed;
    @dawn, dance;
    @evening, sing;
    read (books, $poems, stories) until peaceful;
    study if able;
      write me if-you-please;
sort your feelings, reset goals, seek (friends, family, anyone);
        do*not*die (like this)
        if sin abounds;
keys (hidden), open (locks, doors), tell secrets;
```

```
          do not, I-beg-you, close them, yet.
                                        accept (yourself, changes),
                                        bind (grief, despair);
require truth, goodness if-you-will, each moment;
select (always), length(of-days)
# listen (a perl poem)
# Sharon Hopkins
# rev. 19 June 1995
```

## References

Berry, David M. 2011. *The Philosophy of Software: Code and Mediation in the Digital Age*. London: Palgrave Macmillan.

Borgmann, Albert. 1999. *Holding on to Reality: The Nature of Information at the Turn of the Millennium*. Chicago: University of Chicago Press.

Colombo, Sean. 2016. Steam/C# gamedev: Tatsu—Early Access. Livecoding.tv Webcast. Available online: https://www.livecoding.tv/seancolombo (accessed on 30 July 2016).

Cox, Geoff, Alex McLean, and Adrian Ward. 2004. Coding Praxis: Reconsidering the Aesthetics of Code. In *read_me Software Art and Cultures*. Edited by Olga Goriunova and Alexei Shulgin. Aarhus: Aarhus University Press, pp. 161–74.

Golumbia, David. *The Cultural Logic of Computation*. Cambridge: Harvard University Press.

Hatcher, Ian. 2016a. Drone Pilot Performance. Vimeo Video Recording. Available online: http://ianhatcher.net/#!/video (accessed on 12 August 2016).

Hatcher, Ian. 2016b. *Prosethesis*. New York: Poor Claudia.

Hayles, Katherine N. 2005. *My Mother was a Computer: Digital Subjects and Literary Texts*. Chicago: University Press Chicago.

Hayles, Katherine N. 2012. *How We Think: Digital Media and Contemporary Technogenesis*. Chicago: Chicago University Press.

Heidegger, Martin. 1998. *Pathmarks*. Edited by William McNeil. Cambridge: Cambridge University Press, pp. 239–76.

Hopkins, Sharon. 1995. Listen. *Economist*, July 1.

Ihde, Don, and Evan Selinger. 2003. *Chasing Technoscience: Matrix for Materiality*. Bloomington: Indiana University Press.

Knuth, Donald. 1968. *The Art of Computer Programming: Volume I, Fundamental Algorithms*, Addison Wesley.

Stiegler, Bernard. 2007. *Technics and Time, 2: Disorientation*. Translated by S. Barker. Stanford: Stanford University Press.

TOPLAP. 2016. Transnational, Terrestrial, Transdimensional, Temporal Organisation for the Promotion, Proliferation, Permanence, Purity, Parsimony, Pragmatics of Live AudioVisual, Art, Artistic Programming. Available online: http://toplap.org/about/ (accessed on 30 July 2016).

Verbeek, Peter Paul. 2013. *Moralizing Technology*. Chicago: Chicago University Press.