PROCESS WINDOW
Code Work, Code Aesthetics, Code Poetics

Sandy Baldwin

The Process Window contains general information about the state of the process, with a summary of its current threads and their states.

The odd thing about innovative literature is that no literature is innovative. The familiar but unsolvable paradox of Ezra Pound's rallying cry to "Make it new!" was exactly what made modernist aesthetics so persuasive and productive for the last century of literature. The "new"-ness of poetic innovation comes about against the background of tradition. From this view of the paradox, all novelty re-figures the past. Even if we feel ourselves well beyond modernism, the deep thrill of the new remains in its claim on the future, where each innovation opens a temporal difference within the continuities of literary history. Making it new seems to liven the present with the future. If we are to take Niklas Luhmann seriously, this paradox underlies "art as a social system" (Luhmann 2000, 199–201). The paradox of innovation is that the need to produce new-ness as part of society's self-maintenance is exactly matched by systemic resistance to re-defining the whole, i.e. no innovation can create something actually innovative enough to displace the system. The systematicity of literature, as an institution, is built on this paradox of an innovation that is never more than a repetition.

Integrating literature and information processing in terms of Luhmann's unified systems theory is one possible answer to the paradox. Shannon and Weaver's *Mathematical Theory of Communication* already invoked a concept of literature as exemplifying information density. Luhmann's theory of functional differentiation assumes that society is composed of closed subsystems. Within this theory, making it new is a dynamic maintaining the openness of sub-systems to the environment; for social sys-

tems to accommodate openness, they must internally copy and reflect the distinction between system and environment. Luhmann recognizes that innovative literature assumes this function within the artistic system. One implication is that the auto-telic language focus of contemporary poetries is less a response to a postmodern loss of reference (or something similar) than a self-referential code within a language increasingly employed as an instrumental tool for exchange and commerce. Innovative literature is a meta-code ensuring the stability of the system – in this case, literature – through pure self-reference. This is evident in the popular role of literature: it must produce results that are declared important but are not taken seriously. In this way, the system maintains stability.[1]

The proximity of literature to the root association of poetry as *poiesis* or "making" suggests a more dynamic role than internal self-maintenance. As a regulative meta-code, however, the poetic principle is found today in information processes of coding and re-coding. According to Vilém Flusser's media analysis, every techno-image means a text, i.e., every screen or page is text – markup or op code – written to bring about an image. In response, we produce more writing to explain these techno-images. Techno-images are programs for bringing about texts. Critical texts are plug-ins, continuing the program under the guise of explanation. As a result, we now need to balance the materiality of inscription, and material acts of making or *poiesis* in specific storage media, with the transcoding of text and image, arriving at something like a poetic materiality of transcription. Instead of the "marking" implied by inscription, we are dealing with programs saturated by code without being marked. As a result, the ancient relation of material and form no longer holds: the artist once sought forms within matter; today, the artist channels raw material into machines to create forms through software. Digital media are so many modular devices for directing and forming flows of matter. (Flusser 2002)

The question remains whether the meta-code of innovative literature involves external reference as well as self-reference. No doubt, Luhmann's description is accurate enough, though it does little to explain why literary innovation remains so compelling despite all paradoxes. That is: it works well as a description of "art as a social system" but less well as an account of literature itself. The insistence on systematicity does not solve but merely shifts the paradox of innovative literature. Rather than take this as a failure of Luhmann's rather grandiose theory, it points out the asystematicity of literary innovation. Luhmann's theory offers a displaced version of lit-

erary aesthetics within the rigorous sociological rubric of systems theory. Literature becomes a provisional closure, the institutional site for the introduction and assimilation of innovation. In this account, innovative literature is a medium, a meta-code for observing the dynamics of social systems. Unsurprisingly, this returns us to the paradoxical non-closure of system innovation. The poetic point of systems theory is that innovative literature – as making, *poiesis* – does more than simply thematize the integration of newness into the system: it is what creates the dynamism of the system in the first place.

The following essay attempts to underline the role of this poetic in terms of recent debates about the work of computer code in literary texts. Code appears in the text as a kind of residue or catalyst of machinic processes. The text is "contaminated" by code. In light of the topic of "ergodic poetry," I want to ask what this residue amounts to: what is the work of codework, and how does it relate to other practices of poetry, especially digital poetry? There is no doubt about the fact of this remainder but intense debate about its reading. The debate takes shape around the question of reference: is the fascination with code simply a kind of reflex to an increasingly technological society (codework as aesthetic ideology)? Or is code an external reference to machinic systems and states outside of but "touching" the textual system (code as hermeneutic)? Does codework "instantiate a genuinely 'performative' textuality, a textuality which 'does' something, which alters the behavior of a system" (in John Cayley's critical paraphrase)? My goal is less to arrive at this or that solution than to emphasize the poetic at work, showing that each position displaces but does not do away with the paradox of innovative literature. My starting point is the rallying cry for "electronic space *as a space of poeisis*" in Loss Pequeño Glazier's recent *Digital Poetics: The Making of E-Poetries* (5). For the odd thing about innovative literature is that all literature is innovative. Any given poem will be innovative in purely conventional ways, readable for its experimentation and for its relation to a reasonably stable tradition of experiment. On the other hand, innovation must always remain open and possible. Glazier's central claim bears close attention: it will be through the over-reaching of poetry as the exemplification of digital media, particularly within the current interest in programmable poetry and codework as literature, where innovation shines through as a cultural process within and against literary tradition.

> […] a poem is a large (or small) machine made of words. – William Carlos Williams
>
> […] a computer is nothing but a means for a memory to get from one state to another. – Dr. Jaochim Weyl, opening remarks to the Macy Conference on Self-Organizing Systems
>
> Unreadability of this world / All things doubled – Paul Celan

Must we admit that code is written for the computer, no more and no less? True enough, digital code is compiled into machine instructions to execute on a microprocessor, and the fact that humans can write and read code is purely to ensure that the code be "well-formed" and compilable. Code is "machine-readable," and its appearance for us is a supplement to processes occurring on certain microswitches, invisibly printed below the wavelength of visible light. At the same time, this supplement is more or less a window into the black box. As Florian Cramer puts it in his recent essay "Digital Code and Literary Text," "the namespace of executable instruction code and nonexecutable code is flat" (Cramer 2001). Simply put, this means that the same set of symbols are used in executable code and human-readable text, but Cramer's point lies in the consequences: one cannot tell by looking whether a piece of code is executable or not. In fact, every code is "potentially executable depending on whether there's other code […] capable to process it as machine instructions" (Cramer 2001). Since Cramer extends his definition of code to all text – as coded and subject to algorithms, whether implicit (e.g., grammar) or more explicit (e.g., procedural poetry) – the result is that there is no way within the terms of this argument to distinguish between a given text and executable machine instructions. Every text is the instructions for a possible machine. If all language is formalized and coded, it is equally true that there is no way to circumscribe and stabilize the context of forms and codes. Code purports to be readable by humans and machines, and this presumption accounts for the fascination of the concept "code," offering a hermeneutic of something beyond codes, something machinic and post-human. Cramer echoes Flusser's analyses, where code is defined by "*the possibility to losslessly translate information from one sign system to the other, forth and back, so that the visible, audible or tactile representation of the information becomes arbitrary*" (Cramer 2001, emphasis in original). As a result, Cramer arrives at a radical, anti-material position: there is no such thing as "digital media," despite the many appliances we

all now own. There is only digital information with this or that "arbitrary" material instantiation. The essential translatability of a given code outweighs its internal structure: if code is the extended mapping and binding of tokens against a domain, it must already contain the possibility of remapping against other token-domain bindings. This definition of information determines its qualification as code. Digital code becomes a medium when materialized as an image, a text, a computer, and so on. Code is independent of its hardware. Here Cramer exactly up-ends Friedrich Kittler's influential "media materialism." If Kittler reduces everything to hardware and voltage differentials, Cramer expands software algorithms to concepts that computer programs exemplify without ever exhausting (Kittler 1990).

Cramer adds that it is the fact that one "cannot tell from any piece of code whether it is machine-executable or not," which provide the "principle condition" of "codework." As coined by Alan Sondheim, codework is a flexible designation for a range of artistic practices. Sondheim allows the term codework to cover "just about anything that combines tokens and syntax to represent a domain," but in practice, the work is characterized by the appearance of computer code as part of the text (Sondheim 2001). The contrast is significant: between a broadening of concept to include almost all uses of language and a restriction of practice to works which thematize the definition. Codeworkers explicitly set their work in opposition to writing practices that produce complex multimedia surfaces that conceal or hide the code involved beneath layers of image and text, claiming instead to make code manifest. One way or another, codework is a matter of appearances, of visible residues or catalysts for processes fundamental to the text.

Cayley's recent essay "The Code is not the Text" supplies the critical response to Cramer, showing that the too easy critical assimilation of the display and thematization of code to a "revelation of underlying, perhaps even concealed, structures of control" involves a kind of category mistake (Cayley 2002). The appearance of readable structures of technological control is a mere appearance. And yet, I am interested in defending codework beyond its critical takedown. In its overreaching insistence that something is transcribed, codework names a new poetic moment of innovation and invention. It only names this moment, identifying the momentum of invention within code, but exactly this naming is enough. The insistence that something is transcribed, even in the face of code as fiction or simu-

lacrum, makes evident what was contained and shared within the concept of code all along. In what follows, I discuss Cramer and Cayley as opposing poetic strategies around the paradox of code. Indeed, these arguments are tightly intertwined. Cramer describes the inspiration for his essay in an abstract Cayley wrote for the German "p0es1s" conference. What interested Cramer was Cayley's insistence that one may consider the poetics of digital code in terms of the poetics of literary text without subscribing to the new metaphysics of Friedrich Kittler's "radical post-human reductionism." In turn, Cayley's essay, which grew out of his "p0es1s" presentation, is in part a critique of Cramer's own reductionism. This circularity indicates the complex of issues involved.

Cayley's stated aim is to "disallow a willful confusion of code and text" (2002a). He does not oppose the possible relation of code and text, only a reductive identification of the two. There is no absolute separation, but there are protocols or modes of address to be respected. Cayley's approach analyzes metacritical readings that then draw critical implications; he raises questions of how we move from meta-critical statements to particular examples and practices. It is not that one cannot move from criticism to practice, but rather that such a movement must be supported by "a set of relationships – relationships constituted by artistic practice – between a newly problematized linguistic materiality and represented content" (Cayley 2002a). For Cayley, these relationships characterize innovative literature. Cayley's initial examples, moving from the critical reduction of narratives of code as concept to the reality of digital code itself, situate the critique of codework in the broader context of critical understanding of media technology. He focuses on the referential mix-up between things represented or thematized in language and the things themselves. Nothing could be more commendable and pragmatic as critical practice, especially coming from a poet and thinker who seamlessly combines theoretical insight with poetic invention.

Cayley's argument targets the claim that something is transcribed or contaminated in codework. This insistence "brackets" questions of the "address of specific code segments and texts" (Cayley 2002a). That is, playful integration of code into text overrides possible algorithms or procedures involved in the code. Cayley continues that this bracketing leads to a simplification of the "range of positions of address," so that one is left with a generic notion of code and text – "flat namespace" means nothing else (Cayley 2002a). The flattening of distinctions is a way of extracting an

effect of "contamination" by simplification and a way of bracketing rigorously distinct levels and interrelations between texts into a kind of "literal topography" of shared symbols. By contrast, Cayley would enforce the gaps between levels, where materially identical symbols are processed differently according to their means of addressing. As a kind of rhetorical counter-measure, Cayley provides a list acknowledging these many levels: "machine codes, tokenized codes, low-level languages, high-level languages, scripting languages, macro languages, markup languages, Operating Systems and their scripting language, the Human Computer Interface, the procedural descriptions of software manuals, and a very large number of texts addressed to entirely human concerns" (Cayley 2002a). He adds a footnote indicating the complexity of the HCI as an entire set of levels on its own. Clearly, there seems a marked distinction between scripting language and markup, on the one hand, and assembly code and word processing text, on the other.

Cayley does grant that codework can be understood as a self-referential practice allowing discussion of code as a sign across a range of discourses. The shell of broken code activates the semioticity of the notion of "code." As nothing but shell, this code of code organizes other issues of "identity, gender, subjectivity, technology, technoscience, and the mutating and mutable influence they bring to bear on human lives and on human-human and human-machine relationships" (Cayley 2002a). No doubt, these issues are important but Cayley's point is that none deals with the material specificity of digital code. Cayley's critique appears decisive: the revelation of the truth of digital media offered by the codework aesthetic proves empty. What seemed like revelation is in fact a kind of revelation-effect within the cultural codes of the technoscientific imaginary.

To some degree, Cayley's arguments against the codework aesthetic can be turned from a poetic problem into a debate internal to literary history – into questions of defining the history and framework for the emergence of digital literature, and, consequently, questions of defining what will count as digital literature. Cayley, with reference to Glazier's book, argues strongly for continuity between innovative poetries and digital poetry. These arguments see continuities of poetic method and individual influence, most particularly in relation to the process-oriented poetry of John Cage, Jackson Mac Low and others. In this argument, digital poetry fits within the larger framework of innovative literature. By contrast, the codeworkers seem relatively uninterested with establishing genealogies.

They do not claim alternative genealogies but seek to establish a difference, a break in literary history. More specifically: the codeworkers are interested in establishing a literary avant-garde apart from the L=A=N=G=U=A=G=E aesthetic and tradition that informs Cayley and Glazier. While Cayley and Glazier, to a greater or lesser degree, see digital poetry within the larger movement of innovative poetics, the codeworkers see codework as a new and possibly revolutionary poetics. Or, in a slightly different formulation, codework may involve a new genre alongside an emerging field of digital poetry. This solution replaces the question of the work of code with a question of genre definition. In this case, codework is concerned with the emergence of digital code, whether functional or not, while digital poetry becomes a part of "software art," which requires digital code in order to operate but typically does not make this code visible. Software art may involve straightforward textual processes or dazzling multimedia surfaces, but the aim is to use code to enable artistic production, not to display code. The digital poetry of Glazier, Jim Rosenberg, or Cayley himself, to name only a few from the field that Glazier delineates in his book, offer exemplary instances of electronic poetry integrating software algorithms and techniques to expand the possibility of language art – all without the explicit presentation of code. With this distinction in place, one could then proceed with typologies and internal stylistics for each genre, comfortable with their neat functional differentiation within the artistic system.

Neither of these solutions are particularly persuasive in overcoming the critical impasse over codework. The differences in genealogy and genre prove to be more surface impressions than deep differences. Cramer and Cayley remain remarkably close on many points, not in the least of which is their interest in a critical genealogy which includes Fluxus and the procedural poetics of Cage and Mac Low. Moreover, Cayley is decidedly not against the presentation of code – he even offers some codework of his own as evidence for how to do "codework in the strong sense." The difference lies neither in literary history nor in genre; these themes are symptoms of the more fundamental poetic problem.

Cramer's conclusion bears close attention. He concludes with the declaration that his hypothesis on the nature of digital code is "perfectly verified by codework poetry." He insists on a kind of aesthetic effect literalized in codework: it will "teach us to pay more attention to codes and control structures coded into all language" (Cramer 2001). Cramer as-

serts a force released and become palpable as language in codework poetry. The revelation of this force is in some way adequate yet separate from the force itself – revealed in the medium of language. This force remains hidden in any workable code – it is only revealed in performance – and codework extracts it as writing.

Now, Cayley's target is what he sees as the pseudo-revelation of an immediacy of code and text. His argument tries to give a technical explanation for the performance of "force" but does not explain away the performance itself – Cayley, after all, presents a fascinating sample of his own, a Hypertalk poem/code originally in the earlier essay "Pressing the 'Reveal Code' Key." This "human-readable" text is also "segments of interpretable, working code" (Cayley 2002a). (Or so he says, as there is no way to tell from looking whether the code is compilable or not.) Cayley's point is that his poem adequates code and text, or concept and performance, with no confusion or flattening of levels, creating a codework that can bring about "changes in the body of literature, the literary corpus, both its 'material substrate' and its 'codes of representation'" (Cayley 2002a). So, this is "codework in the strong sense."

Cayley's careful correlating and enumerating of distinct levels and strata of meaning reassuringly controls what is revealed, but the paradox of codework is that the effect of revelation occurs even without such claims for precision. The execution of code over-reaches any physical explanation. For Cramer, the codework practitioners prove their point all the more in frequently working with "plain ASCII text" rather than hypertext or multimedia. Not only does codework employ ASCII text with no software or plug-ins; not only does it employ non-functional or broken code, which may be extracted and edited from a context where it did once work; but codework may also employ invented code, fictional constructs presenting a kind of simulacra of code.[2] In fact, this fictionality is central to Sondheim's definition of codework. Here, Cayley's question of address and reference is overreached to the point of absurdity: not only is codework no longer code but also it may never have been code – it may never have worked. Cramer drives the point home: "The contradiction between a complex techno-poetical reflection and low-tech communication is only a seeming one; quite on the contrary, the low-tech is crucial to the critical implication of the codework poetics" (Cramer 2001). But what is the critical implication of the low-tech?

The answer seems to lie in Cramer's definition of code algorithms, elaborated across several essays. The empty revelation of code reflects the execution of an algorithm that exceeds any possible physical performance. Cramer's favorite example of an algorithm is La Monte Young's Fluxus piece consisting only of the instruction to "Draw a straight line and follow it" (Cramer 2001). Cramer points out that "the instruction is unambiguous enough to be executed by a machine" while at the same time "thorough execution is physically impossible" (Cramer 2001). For Cramer, this example generalizes to all algorithms: there are only failed implementations of algorithms. In any particular execution of a code, the algorithm remains conceptual and mental. Cramer adds that if "such implications lurk in code, a formal analysis is not enough" (Cramer 2001). No amount of formal analysis will adequate the structure of software to its cultural forms.

Codework addresses this paradox through an "aesthetic extremism" (Cramer 2001). Cramer's definition of an algorithm sees no difference between code executed on the computer or printed in a book, or, for that matter, "executed in the mind of the reader" (Cramer 2001). The work of code is in the reader or observer, not in the physical kinetics of this or that screen or poem. Codework teaches us to "pay attention" or more frequently leads us to "reflection." The work of reflection completes the work of codework, and the broken fictionality of codework only increases this reflexivity. The failure of the algorithm in its contingent materiality is its success in concept, a success that is experienced and read but in no way visible in the text. The fictionality of codework is the guarantee of this experience. In this sense, Cayley is absolutely right: the codework aesthetic is literally meta-physical, since it implies a movement beyond any physical movement.

The aesthetic reflection produced by codework is the concept of the code algorithm stripped of its contingent materiality. The fictionality of codework – its innovation – is central here. As fiction, codework lays bare our fascination with code as external reference, as genuine performance. It is the performance of this fascination that codework extracts. One is left with a kind of "hyper-reflection," in the sense alluded to by Maurice Merleau-Ponty in his posthumous writing, a blindspot in consciousness containing the invisible infrastructures of perception. Thus: codework as hyper-reflection on the reflexivity of all codes, on the inaccessibility and distance between code and text, and on the conditional opening of exactly the levels of coding Cayley invokes.

There is little point here in dispelling the metaphysical presuppositions of the codework aesthetic – what is "force" after all? – since both sides of the argument take it for granted. This aesthetic is central to our notions of digital poetry. Jim Rosenberg's hypertext poems, invoked by Glazier as "one of the most valuable investigations currently underway" in digital poetry (Glazier 2002, 137), are written in terms of a similarly impossible conceptuality and contingent materiality. Rosenberg defines hypertext as a way of representing a network that could be represented by "other means than using a computer – on paper, for instance" (Rosenberg 1996). At the same time, Rosenberg takes hypertext as a way of thinking not yet possible in any given technology. That is, hypertext is an approach to poetry first and only secondarily a function of technology.

Rosenberg's poems are "simultaneities": piles of words, stacked clusters of word "skeins," following his insight that such juxtaposition is " the most basic structural act" (Rosenberg 1996). Mousing over "opens" the simultaneity to reveal an individual skein, a scatter of words and phrases, with "vertical" relations indicated by changes in font. The simultaneity is a poem that emits readable texts. Each text is the outcome of the user's mouse interactions with Rosenberg's programmed relations between skeins. Appearances are conditioned by the user's attention via the structure perception-mouseover-poem. The poem is an opening. While it is possible to speak of particular textual conditions enabling Rosenberg's work – the tradition of Mac Low's simultaneities, the availability of easily programmability HyperCard stacks, etc. – none of these adequately accounts for what happens as individual skeins appear and disappear. The simultaneity remains in a kind of quasi-space and -time prior to the text. Mousing over is the real time of the poem. The resulting words are not inscriptions but transcriptions of the user's movement and attention, all within the conceptual algorithm of the poem. As a "fundamental micromaneuver at the heart of all abstraction," the simultaneity produces a minimal possible world, a phenomenology of momentary objects (Rosenberg 1996). Rosenberg argues that we should try to think of hypertext as "a medium in which one thinks 'natively'" (Rosenberg 1996).[3] The paradoxical task is to think of the technology that would be adequate to "an individual thought" that "is entirely hypertext" (Rosenberg 1996). Rosenberg writes a poem for technology not yet available. In this disjunction of grand conceptual apparatus with its instantiation in digital media, Rosenberg's work is innovative by means of its own failure – and this recognition is in no way intended to

mean that Rosenberg's important project is a failure. These poems mark the structural relation between a poem and itself as an act of innovation. The poem is innovation's "*mode of disappearance*," as Jean Baudrillard puts it (1993, 213).[4]

The poetic code, particularly within the systematicity of literature and digital media, is not a simple reflection or afterimage of technical functions but the opposite: the complexity of the concept of "code" is a metaphor for poetic innovation. The drive towards a poetics of code, focusing on systems of constraint between natural languages and artificial languages, is built on the absent point of reference between code and text. It matters little whether this point pivots around the precision of different levels of coding or on the translatability of codes (between strong or weak codework). The point is rather to read the irreducible poetic invention that enables systemic cultural reflection, "to read what was never written," in Walter Benjamin's words (1978, 226). "Code as writing" means that the singularity of poetic invention provides the mediation needed to conceptualize information. Systems theory, and its extension to all information exchange, is a metaphoric explanation of what poetic innovation brings about.

NOTES

1. Compare Luhmann's analysis of "the modernity of sciences" (2002, 61–75).
2. Cayley is explicit that his target is critical discussion of codework and not codeworkers themselves, and carefully brackets several of the most prominent of these writers, showing that his critique does not apply in every case. At the same time, the practice of contaminated and invented languages, best characterized by MEZ's "mezangelle" is clearly the most problematic codework practice for Cayley.
3. Editor's note: Rosenberg uses "natively" here metaphorically, from the technical usage where, for example, code is "native" to a particular operating system, i.e. system and code are designed for one another, with no additional or mediating encoding or compilation required.
4. This concluding chapter of Baudrillard's *Symbolic Exchange and Death* is a valuable contribution to analysis of poetry, often overlooked for his more well-known essays on simulation.