THE LIVE CODING OF SLUB
- ART ORIENTED PROGRAMMING AS MEDIA CRITIQUE

Christian Ulrik Andersen
Ph.D., Assistant Professor
Information and Media Studies
University of Aarhus, Denmark
cua@multimedia.au.dk

ABSTRACT: Computer art is often associated with computer-generated expressions (digital audio/images in music, video, stage design, etc.). In recent computer art, however, the code-text itself – not the generated output – has become the artwork (Perl Poetry, ASCII Art, obfuscated code, etc.). This paper will analyze and discuss code as the artist's material. In particular, it will focus one particular artistic code-praxis: the Live Coding performances of Slub (programming computer music live, visually showing the coding)

The artists Alex McLean and Adrian Ward (aka Slub) along with Geoff Cox declare: "art-oriented programming needs to acknowledge the conditions of its own making – its poesis." (Cox et. al. 2004) The paper will argue that this statement formulates a media critique. As Florian Cramer has proposed, the GUI represents a media separation (of text/code and image) causing alienation to the computer's materiality/text. (Cramer 2003) The paper will then propose that object of art oriented programming – in an avant-garde perspective – must be to recuperate an interchangeability of data and processing. How?

The particularity of Live Coding does not rely on the magical expression – but nor does it rely on the code/material/text itself (as one might say is the case in some code-art). It relies on the nature of code to do something – as if it was magic: in the performative aspect of the code. Relying on performance theory (Austin, Carlson) the paper will demonstrate how the computer in the Live Coding sessions is much more than mere mechanic performance. The paper will explain how code itself is staged as performative language (interchanging data and process) and further focus on the performance of code before an audience. Arguing that the performance excludes the audience (esoteric code), the paper will raise the question of whether listening to the code (at a non-textual level) may provide an embodied experience of data-processing. The performance can be regarded as a collective appreciation of the code artist as a musician using code as his/her instrument, interchanging data and process live.

An interest in code has been a central theme in computer art through the past decade or so. Amongst many 'software artists' there is a fundamental interest in creating expressions that consider and play with aspects of the computer code – such as poetry written in the programming language Perl ('Perl Poetry' – as displayed here). Even the 2003 edition of Ars Electronica was entitled 'Code – The Language of Our Time'. This aesthetic interest in code language is of course not new – but has been vigorously investigated in many different ways –exchanging the order of the book with the chance of the dice by Mallarmé; addressing the reader using instructions (as by Tristan Tzara, displayed here); in the concrete poetry of the 50s; in the 'psychogeography' of the situationists; in avant-garde music by John Cage – and so forth.

One of the most interesting examples of today's 'code art' is the *Live Coding* performances as for instance performed by the artists Alex McLean and Adrian Ward

(aka *Slub* and I mention them because McLean and Ward, along with Geoff Cox, have been very explicit about their praxis – but in general one can consult the website toplap.org for an overview of the praxis of live coding: events, writings, software, etc.).[1]

What is unique in a live coding performance is the incorporation of a *live* aspect of code, not usually seen. In their performance the two programming artists experiment with live programming of electronic music. Instead of having pre-programmed the music or performing through a graphical software application interface (a sequencer e.g.), the live coders appear as musicians mastering the code as a musical instrument. Through large screen projections displaying the artist's code, the audience at their side gets a glimpse of the code as the creative force in the music. To emphasize the particularity of the live coding performance, the code-text is ultimately deleted: Contrary to how we usually see computer code, the code is ephemeral; it is the coding performance that is the work of art – not the algorithms themselves.

How do we perceive this obsession with code in software art? What makes a 'code performance' particular – to the user of a computer; to the programmer; and to the audience?

In this presentation I want to discuss *live coding* as a media critique; the live coders' reasons for reclaiming what could be called a *code poesis*; and finally, what all this means to an ignorant audience. I will argue that we must understand the live coders' obsession of code as an affirmation of a performative aspect, usually repressed in our operation of the computer. Their musical performance involves a critique of the computer as media and of the status of both programmer and user in our society.[2]

Code as media critique

> When Alan Kay developed the first graphical mouse-controlled computer environment at Xerox PARC in the

[1] Performed e.g. at the Read_me festival for software art in Aarhus, Denmark, 2004. http://projects.dorkbot.org/rd04/wiki/MediaFiles?action=AttachFile&do=view&target=livecoding-read_me04-01.mov
Many other audio and video examples can be found at http://toplap.org.

[2] In my discussion of the roles of the user and the programmer, I very much depend on the aesthetic and political philosophy of Jacques Rancière – but, because of the scope of the conference being 'code' and the limited period of time we are granted as speakers, I feel obliged to keep this reference implicit.

> 1970s, the separation between ‘usage’ and ‘programming’ was for the first time implemented as separation of media. ‘Usage’ became graphical, ‘programming’ textual. The gap widened with the commercialization of Kay’s ideas through the Apple Macintosh and Microsoft Windows.
> (Florian Cramer 2003, 100)

As interface historians often have stated, it is the invention of the Graphical User Interface that made the computer available to a general public. The German software theorist Florian Cramer, however, describes the negative effect of the GUI as an absolute media separation (Cramer 2003, p. 100). The GUI seemingly separates the computer code (the text) from the use of the computer (the image). Before the GUI, this was not the case. In the old days – where you interacted with computers by writing instructive text– the one using the computer and creating its images was also the one writing the text/programming the machine. When the user no longer writes the text and the code becomes invisible, her role is reduced to the one of the reader.

One may actually say that ‘the user’ as a passive, consuming recipient is born the moment you separate the text from the image. From having been the user’s own machine that she coded herself, the user is now staged as a spectator – with very limited awareness and control of her own position. The computer is absorbed by the consumer society.

An artistic uproar against the invisibility of the code is, as such, a resistance to a visual, capitalistic, consumer, media society. This revolt is not unlike a traditional avant-garde way of thinking and acting: It wants to reveal the consumer society’s anaesthetization and alienation (‘verfremdung’) of the consumer. The hacker, the one insisting on accessing the code and writing her own code, does to the graphical user interface what Brecht did to the theatre. With a reference to Roland Barthes’ idea of ‘le scriptible’ (Roland Barthes 1970, p. 10), Cramer thus advocates for a ‘writerly computing’ where the user is not just a consumer but also a producer of the text (Cramer 2003, p. 10). In this line of thinking, the user should once again become a programmer.

A broad line of software artists claim – partially inspired by Cramer – that in a culture where our most valuable tools are computer interfaces that organize and represent invisible data as images, one should maintain a critical approach to these

representations. The present trend in computer art, exhibiting the otherwise revealed code, perceives itself as such a critique.

The *Live Coding* performances however, I believe, cut deeper into the implications of this so called 'media separation' alienating the user. They not only suggest that the code itself is suppressed, they also suggest that a *performative* aspect is suppressed in the usual employment of the computer in our society. To understand this and how and why *live coding* revives a performative aspect of code, we must first look deeper into how computers and code actually operate.

The role of the reader/user: The performative language – code as instruction

> "Open, sesame!"
> (1001 Arabian Nights)

The critique of the Graphical User Interface seems to favour the text for the graphical. The issue, however, is not whether the presentation of the computer is graphical or textual. It is a question of semantics and grammar. We must therefore begin by taking a closer look at the particularity of code as a type of language.

John Langshaw Austin states in *How to Do Things With Words* that language does not just make sense by referring to- or describing an object in a factual way.

Most likely, nothing would ever happen if someone incidentally stood in front of the cave of the 40 thieves and said "cave". What is crucial is that Ali Baba knew 'the magic words'. We are witnessing two oppositional language systems: An abstract, symbolic language with signs relating to objects (as when I say "cave") and a performative, magic language where speaking involves an action (as when I say "open, sesame!").

Austin draws attention to these other semantic sides of language and speech not usually emphasized. The utterance of certain sounds may refer to a meaning. But apart from its referential signification it is also characterized by *il-locutionary* acts. In saying something you actually also *do* something. You may for instance promise something, order something or claim something. What is meant, of course, depends on the situation – a semantic, discoursive level of language, allowing the user of language to generate more complex enunciations.

Even though computer code is not meant to be spoken, code as a type of language compares very well to Austin's ideas of 'speech acts' (as also German curator and media theorist Inke Arns confirms).[3] Computer code is full of 'magic words' and it is essentially performative. This implies a language that is instructive and a language where the situation dictates the meaning of the speech act. This way, by referring not to the meaning of the word but to the situation in which it is used, higher levels of complexity can be reached.

Not all ways of instructing a computer, however, allows a semantic complexity. At an abstract level where users interact with computers, we find the graphical user interface. The graphical user interface tends to be predominantly referential. Symbols refer to certain meanings and actions. The 'trash bin' symbol signifies the deposition of files e.g. The one using the graphical symbols to operate the computer is thus cut off from semantic operations where you combine words and create situations yourself.

Whereas the average user of a computer operating graphical symbols uses a referential language, the *live coder* insists on using the code as a speech act. Exhibiting this type of interaction based on the programmer's own algorithms and instructions is not so much about denying the visual images of the computer and promoting its textual level as it is about stressing a certain grammar and semantics of the computer that the graphical user interface traditionally denies and suppress. What is rejected is not the images themselves but the fixation of the user of the images and the assertion of other behaviours and languages that escapes these traps of visual culture. In that sense, live coding carries several characteristics of other contemporary art genres also reclaiming a performative aspect of language and a political, democratic right to express oneself.

But it is not just the use of the computer and the role of the user that is transformed in live coding; it is also the programming and the role of the programmer.

---

[3] As discussed at SLSA'07 conference*, Code*, in Portland, Maine, Austin's speech act theory seems only appropriate when talking about code as a bodily, sensual experience – as it is the case in Live Coding. Code as perceived and performed by the computer is better described by linguistic theories focusing on the performative properties of language itself, isolated from the body (as e.g. the semantic theories of Émile Benveniste).

The role of the author/programmer: The performing machine – code as ephemeral

> *... code as performative: that which both perform and is performed.*
> (Cox, McLean & Ward 2004, p. 161)

In the 1930s the British mathematician Alan Turing proposed a sort of calculating machine where a mathematical language based on numbers (arithmetic) is replaced with a mathematical language based on letters (algorithms). This machine can perform any mathematical calculation – as long at it is represented as an algorithm, meaning a well defined (textual) instruction for alterations of states. Perhaps one could say that this machine holds not the calculation but the recipe of the calculation.

The Turing-machine (so called – and, later to be known as the computer) is thus characterized, not by the calculations as such, but by their symbolic expression and the possibility of making the symbolic expression available in a physical/mechanical form. A computer is therefore a particular technological invention characterized by being (both) text *and* machine, media *and* tool – an invention where information and process are inseparable.

As such, one may talk about a computer as a tool, a machine that performs – just as other machines perform. This type of performance however – contrary to the raw power of the engine – makes little sense without understanding that that which is performed (the 'recipe' or the text based instruction) is also that which performs. One does not appropriate the computer as a tool without acknowledging the text-media. *Live Coding* seems to stress this (lost) unification of media and machine by working with and displaying code as *live*. In *Live Coding* an important aspect is the insistence on a type of employment of the computer where the code at once processes (appear on the screen as instructions for the machine) and is processed (appear as music).

Hence, *Live Coding* is not about finding an algorithm 'that sounds beautiful' (the beauty of mathematics and machines) but about interacting with the machine, constantly making alterations and development in the music. The code is not just a set of instructions – it has become ephemeral. The point of writing code in the live coding sessions is not just to leave a set of instructions for the computer but to use instructions to manipulate the music, approaching a situation where they operate the computer as a musical instrument. As the live coders express it themselves:

> ”Live coding is the activity of writing (parts of) a program while it runs. It thus deeply connects algorithmic causality with the perceived outcome and by deconstructing the idea of the temporal dichotomy of tool and product it allows code to be brought into play and artistic process.”
> (Alexander et. al. 2004, p. 244)

Live Coding, thus, stands in sharp contrast to the usual experience of the programmer - producing text and then feeding the text to a computer that produces something. He is now a performing artist in search for an experience of a *differance*, a conceptual gap between instruction and outcome – an experience of a *code poesis* (the term used by the live coders themselves). He is searching for *the making* of computer music:

> ... art-oriented programming needs to acknowledge the conditions of its own making – its poesis.
> (Cox, McLean & Ward 2004, p. 161)

Now, having dealt with how *Live Coding* challenges the roles of both computer users and programmers, one must acknowledge the fact that the audience at a live coding concert might not even understand code or care about the programmer. They might just be there for the dancing. In the coding sessions it is not just a computer performing music or a programmer writing code. A performance is always a performance for *someone* and live coding music performances are live concerts before an audience that sees the code but don’t necessarily understand it.

The role of the spectator: The musical performance – the agency of code //esoteric code

> ”It is not necessary for a lay audience to understand the code to appreciate it, much as it is not necessary to know how to play guitar to appreciate watching a guitar performance.”
> (Alexander et. al. 2004, p. 248)

My last and concluding remark on the live coding performances is then that writing code must be seen as agency - *a way of acting*.

Only to the machine and the initiated does code make any sense. The code and the writing of code involves, however, a third kind of performative action. In Austin’s terms, this would be a *perlocutive* act that, when performed with the machine, *does*

something to an audience. *Live coding* is not an attempt to make people understand the exhibited code but an attempt to affect the audience – and possibly to make them dance.

By transferring his experience as a programmer from someone who writes instructions to make the computer act, to someone who actually acts himself and manages to use the code to create music that may be appreciated by an audience, the programmer attempts to redistribute his sensual experiences – in the terms of Jacques Rancière.[4] The object of *live coding* is to direct the conditions of creation towards a *poesis* of code – a kind of making, directed towards an aesthetic practice of expressing oneself. This experience and investigation of a code *poesis,* that forms the basis of using the computer as a personal medium for expression, is something that is usually not permitted in computing. In computing users are staged as actors who handle predetermined signification, and programmers are merely fulfilling the need to make new features available to the users. Applying a creative, semantic, complex use of computer language, transforms the computer into a means of expression. Using code in an aesthetic practice may sound good and may make people dance but it is therefore essentially also a political action insisting on the right to speak and on one's own control over technology. Admiring such attempts does not necessarily involve an admiration of the computer's performance, nor of the code itself but of the programmer's incisive insistence of using the computer and the code in his or her own way. The programmer is transformed from a technician to a wizard whose mastery of the magic words and redistribution of the senses, we may all admire – even though we might not understand them.

---

[4] *"I conceive it [the wild appropriation of high language by the common people in the 19th century] as the widespread availability of writing which meant the very condition for making history: the possibility for anybody to appropriate for him- or herself another ethos than the ethos suited to their condition."* (Jacques Rancière 2005: 16).

Rancière's conception of a redistribution of the sensual experiences corresponds to an appropriation of an 'ethos' (experiences, thoughts and behaviours) not normally assumed by a class – and often forbidden and illegal. In this quote Rancière refers to the act of writing poetry (the language of the bourgeoisie) and the establishment of nocturnal literary societies by the working class in 19th century France. Surprisingly, he does not see the class struggle as a complaint about the distribution of goods. What is at stake is the challenge of a social hierarchy's determination of life. The worker, who in a platonic world view ought to work at day and sleep at night, 'redistributes the sensible' by changing behaviour and challenging the norm – by doing what is not authorized: make noise, adopt a different (high) language, create nocturnal poetry societies, make newspapers, etc. My basic assumption is that this type of behaviour corresponds very well to some cultural groups' appropriation of the computer code today (the hacker e.g.).

Literature:

- Arns, Inke (2005): ”Read_me, run_me, execute_me – Code as Executable Text: Software Art and its Focus on Program Code as Performative Text”. Online (11/4/06): http://www.medienkunstnetz.de/themes/generative-tools/read_me/scroll/
- Alexander, Amy (et. al.) (2004): “Live Algorithim Programming and a Temporary Organisation for Its Promotion” in Goriunova, Olga & Shulgin, Alexei (red.): *read_me – Software Arts and Cultures, Edition 2004*. Århus, 2004, pp. 160-175.
- Austin, J. L. (1955): *How To Do Things With Words – The William James lectures delivered at Harvard University in 1955*. Oxford University Press, 1962.
- Barthes, Roland (1970): *S/Z*, Éditions du Seuil, Paris 1970.
- Carlson, Marvin (1996): *Performance. A Critical Introduction*. Routledge, 1996
- Cox, Geoff; McLean, Alex & Ward, Adrian (2004): “Coding Praxis: Reconsidering the Aesthetics of Code” in Goriunova, Olga & Shulgin, Alexei (red.): *read_me – Software Arts and Cultures, Edition 2004*. Århus, 2004, pp. 160-175.
- Cramer, Florian (2003): ”Exe.cut[up]able statements: The Insistence of Code” in Stocker, Gerfried & Schöpf, Christine (red): *Code – The Language of Our Time*, Ars Electronica, Linz: Hatje Cantz, 2003, pp. 98-103.
- Rancière, Jacques: ”From Politics to Aesthetics”*,* in *Paragraph,* Mar 2005, Vol. 28 Issue 1, pp. 13-25.
- Turing, A. M. (1937): ”On Computable Numbers, with an Application to the Entscheidungsproblem” in: *Proceedings of the London Mathematical Society 42*, pp. 230-265, 1937. Re-print in *The Undecidable* (red. M. David). Hewlett, NY: Raven Press, 1965.